Actas de las Jenui, vol. 1. 2016.
Páginas: 297-300

Enseñanza de sistemas empotrados: De Arduino a Raspberry Pi

Carlos Catalán Cantero
Dpto. Informática e Ingeniería de Sistemas
Universidad de Zaragoza
E.U. Politécnica de Teruel
ccatalan@unizar.es

Alfonso Blesa Gascón
Dpto. de Ingeniería Electrónica y Comunicaciones
Universidad de Zaragoza
E.U. Politécnica de Teruel
ablesa@unizar.es

Resumen

Paradigmas como *Cyber-physical Systems* (CPS) e *Internet of Things* (IoT) han aumentado la importancia de los sistemas empotrados. En muchos casos estos requieren el uso de completos computadores en vez de sencillos computadores basados en microcontroladores. Esto aumenta el nivel de complejidad de las asignaturas que imparten dicha materia, especialmente para alumnos con menores conocimientos de programación. En este trabajo se propone un modelo de programación para Raspberry Pi basado en Arduino, que ha empezado a utilizarse durante este curso, con el fin de disminuir en parte la indicada complejidad.

Abstract

Paradigms such as Cyber-Physical Systems (CPS) and Internet of Things (IoT) have increased the importance of embedded systems. In many cases these require complete computers instead of simple microcontroller-based computers. This increases the complexity level of the subjects on the matter, especially for students with lower programming skills. This paper presents a programming model based on the Arduino for Raspberry Pi platform that it has been used this year, in order to hide some of this complexity.

Palabras clave

Sistemas empotrados, *Cyber-physical Systems*, *Internet of Things*, Arduino, Raspberry Pi.

1. Motivación

La enseñanza de sistemas empotrados es cada vez más importante dentro de la formación de los futuros ingenieros en informática, automática o electrónica; especialmente si se enmarcan en los paradigmas *Cyber-physical Systems* (CPS) [11] e *Internet of Things* (IoT) [9]. Los CPS representan una nueva generación de sistemas empotrados que deben controlar diferentes objetos y sistemas de manera escalable, distribuida y descentralizada. En ellos se hace particular énfasis en los aspectos de comunicación mediante los protocolos estándar de TCP/IP.

Respecto de los aspectos computacionales estos nuevos sistemas empotrados deben tener capacidad multitarea. Este requisito viene dado por la necesidad de atender la comunicación con sensores/actuadores, en muchos casos a través de buses (e.g. I2C), con otros dispositivos y con la interfaz de usuario, así como de la ejecución con restricciones de tiempo real.

Estos aspectos han aumentado la complejidad de los sistemas empotrados. A este respecto, cada vez es más adecuado el empleo de plataformas hardware previamente desarrolladas. Un primer paso son aquellas con microcontroladores y sin sistema operativo como Arduino (http://www.arduino.cc). Pero dicha complejidad exige el uso de plataformas más potentes. Por ejemplo, computadores *System on Chip* (SoC) dotados de un sistema operativo (e.g. *linux-like*). En este caso, el alumno debe diseñar el software de aplicación y de comunicación con otros posibles dispositivos, así como con sensores y actuadores. Para ello, se apoya en servicios ofrecidos por el sistema operativo, en librerías y, ocasionalmente, en *runtimes* de terceros.

Existen desde hace tiempo computadores SoC comerciales, aunque hasta hace poco su elevado precio ha dificultado su empleo en la enseñanza. El uso masivo de estos sistemas, particularmente en dispositivos móviles, ha hecho que dicho precio se haya reducido en gran medida. Gracias a ello han surgido computadores SoC de bajo coste, desde el punto de vista docente un ejemplo relevante es Raspberry Pi (http://www.raspberrypi.org).

Los sistemas empotrados es una materia que se presta bien a asignaturas orientadas a proyectos, donde los alumnos deben construir "artefactos" en vez de realizar prácticas de laboratorio más o menos guionizadas.

El desarrollo de estos proyectos puede ser difícil, especialmente en alumnos con carencias en programación multitarea o en red [7]. Incluso en aquellos con dichos conocimientos el tiempo de desarrollo, un recurso siempre escaso en cualquier asignatura, puede ser grande.

En este trabajo proponemos un modelo de programación sobre Raspberry Pi definido y diseñado para una asignatura de sistemas empotrados. Dicho modelo se inspira en el modelo de programación de Arduino, el cual extiende al objeto de facilitar la programación multitarea y la comunicación entre dispositivos. Un *framework* facilitado al alumno es el encargado de construir y ejecutar las aplicaciones.

En el Apartado 2 se discuten brevemente las principales ventajas e inconvenientes de Arduino y Rasbperry Pi, desde el punto de vista de la enseñanza de sistemas empotrados. Por su parte, en el Apartado 3 se presenta el modelo de programación propuesto y su *framework*, mientras que en los Apartados 4 y 5 se describe y analiza, mediante un caso de estudio, la experiencia docente llevada a cabo. Finalmente, se destacan las conclusiones más importantes del trabajo.

2. Arduino vs. Rasbperry Pi

El empleo de estas plataformas en la enseñanza en ingeniería ha sido estudiado en diversos trabajos (e.g. [4, 10, 3]). Además de su bajo coste, su principal ventaja es tener una gran comunidad de usuarios detrás. Por lo que hay disponibles multitud de tutoriales, herramientas, ejemplos de uso, librerías y hardware de E/S. En el caso particular de Arduino las aplicaciones pueden ser escritas en cualquier lenguaje de programación disponible para su microcontrolador AVR. Aunque, existe un IDE *open source* (`http://www.arduino.cc/en/Main/Software`) para el desarrollo de programas, llamados "*sketches*", en C/C++.

Un *sketch* tiene dos funciones denominadas: `setup` y `loop`. La primera es ejecutada una sola vez al principio para acciones de inicialización y configuración. Por su parte, la segunda se ejecuta a continuación repetidamente de manera indefinida. Este modelo de programación monotarea facilita el desarrollo de aplicaciones sencillas.

Existen diversas propuestas para incorporar la capacidad multitarea a Arduino, pero algunas son incompletas (e.g. [5]) y el resto están limitadas por las prestaciones de su hardware (e.g. [8]). Por su parte, la comunicación con TCP/IP es posible también mediante módulos externos accesibles vía SPI o UART del microcontrolador. En cualquier caso, es difícil cumplir con los requisitos de muchos sistemas empotrados.

Por su parte, Raspberry Pi tiene un hardware mucho más potente que puede ejecutar una distribución ligera de Linux derivada de Debian (i.e. Raspbian). Ello facilita la programación clásica de aplicaciones multiarea con *threads*, así como la comunicación TCP/IP mediante *sockets*. Todo ello, utilizando las herramientas de desarrollo estándar de Linux. Además, aunque Raspbian no es un sistema operativo de tiempo real puede llegar a serlo si es "parcheado" (e.g. PREEMPTIVE_RT, Xenomai). Las mencionadas características de esta plataforma exigen del alumno un nivel mayor de conocimientos de programación.

```
...
@public(const,persistent) int foo1 = 0;
int foo2 = 0;
@remote() int foo3;
...
void setup() {
  ...
}
void loop1(int period = 20) {
  @public() double foo3 = 0;
  ...
  @synchronized(mutex1)
  ...
  @end
  ...
}
void loop2(int period = 500) {
  ...
  @synchronized(mutex1)
  ...
  @end
  @get("disp2.foo3")
  ...
}
void finalize() {
  ...
}
```

Figura 1: Esquema del código de una aplicación.

3. Modelo de programación para sistemas empotrados

Una aplicación desarrollada con este modelo se compone de las siguientes funciones (ver Fig. 1):

- `setup()`: Mismas características que Arduino.
- `finalize()`: Ejecutada al finalizar la ejecución para desactivar salidas, liberar recursos, etc.
- `loop<n>(int period = <ms>)`: Mismas características que Arduino. Puede haber más de una de estas funciones (`loop1`, `loop2`, etc.), ejecutadas concurrentemente con un prioridad establecida de forma automática por el *framework*, según su periodo con el esquema *rate monotonic*. Cada función es ejecutada mediante una tarea de tiempo real, teniendo que ser el conjunto de estas planificable.

El modelo usa anotaciones para abstraer los detalles de implementación de: a) el acceso concurrente a recursos compartidos (e.g. variables) y b) la comunicación entre dispositivos mediante el acceso a variables de la aplicación. Estas anotaciones son:

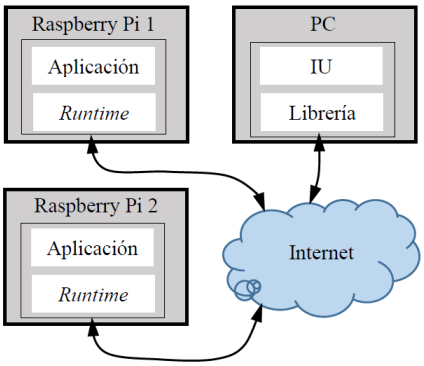

Figura 2: Ejemplo de aplicación.

- `@synchronized()` y `@end`: Indican el comienzo y final de una sección crítica. Un parámetro de la primera indica el *mutex* de la sección crítica.
- `@public()`: Indica que una variable es accesible desde un dispositivo externo. Esta anotación puede incluir como parámetros `const` y/o `persistent`. El primero indica que la variable es de solo lectura para los dispositivos. Por su parte, el segundo indica que su valor será persistido por el *framework* al finalizar la ejecución actual y recuperado antes de empezar la siguiente.
- `@remote()`: Indica que una variable está declarada en un dispositivo externo.
- `@get()` y `@set()`: Accede y modifica respectivamente una variable remota.

3.1. *Framework*

El *framework* incluye un preprocesador que traduce el código de aplicación con funciones y anotaciones en un módulo C estándar. Este módulo es compilado y enlazado con un *runtime* para generar el programa ejecutable. El preprocesamiento y compilación están automatizados mediante la herramienta *make*. El *runtime* realiza también la comunicación entre dispositivos e interfaces de usuario remotas (ver Fig. 2). Se incluye una librería Java para el desarrollo de estas últimas que oculta detalles de comunicación entre dispositivos.

4. Caso de estudio

Raspberry Pi, el modelo de programación y el *framework* descritos en los anteriores apartados se han empezado a utilizar durante este curso en una asignatura de sistemas empotrados, de un grado de ingeniería automática, que sigue una metodología basada en proyectos. Se han propuesto distintos proyectos, entre los cuales está el caso de estudio del sistema empotrado de una sencilla lavadora automática, cuyas características de manera breve son:

- Acceso a sensores y actuadores mediante los puertos GPIO y el bus de comunicación I2C.
- Automatización de varias secuencias de lavado.
- Interfaz de usuario con pulsadores y display LCD.
- Supervisión y monitorización remota.

Los alumnos han podido introducir otras características que mejoren el resultado final del proyecto, por ejemplo, minimizar el consumo eléctrico.

A partir de estas características los alumnos han desarrollado sus diseños, los cuales han sido analizados en clase estudiando sus ventajas e inconvenientes, haciendo especial énfasis en cómo el diseño hardware afecta al diseño software. Por ejemplo, el acceso de algunos sensores vía bus I2C exige una tarea periódica específica, debiendo haber otra distinta para los actuadores (e.g. PWM). En la Fig. 3 se muestra un posible diseño hardware.

A la hora de poner en marcha y supervisar el sistema los alumnos pueden utilizar un sencillo programa, facilitado previamente y ejecutado en un PC, a través del cual pueden consultar y modificar remotamente el valor de variables de la aplicación públicas. En el próximo curso se contemplarán proyectos que requieran la comunicación entre varios sistemas.

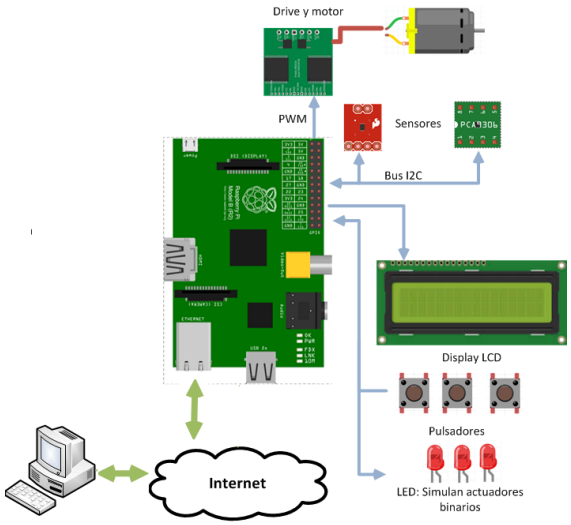

Figura 3: Diseño del hardware del caso de estudio.

A los alumnos se les ha facilitado el esquema del código de una aplicación (ver Fig. 1). El sistema a controlar requiere la ejecución de varias funciones `loop` con diferente periodo, que en algunos casos deben acceder a variables compartidas. Un ejemplo de conjunto de funciones para el caso de estudio propuesto es el mostrado en el Cuadro 1. Para el correcto funcionamiento del sistema el conjunto de tareas asociadas debe ser planificable, algo que puede determinarse, por ejemplo, mediante la sencilla ecuación de Liu y Layand [6].

Función	Periodo(ms)	
`loop1`	2	Generación de PWM
`loop2`	20	Atención a GPIO e I2C
`loop3`	100	Control/automatización
`loop4`	200	Interfaz de usuario

Cuadro 1: Funciones `loop` del caso de estudio.

5. Breve análisis de la experiencia

El análisis de la experiencia realizada con la propuesta es necesariamente preliminar, debido a ser el primer curso que se han empleado este modelo de programación y el *framework*, aunque pueden extraerse algunos resultados. El primero es confirmar la facilidad de desarrollo con ellos de aplicaciones multitarea para alumnos "no-informáticos". Por otra parte, el hecho de utilizar una plataforma como Raspberry Pi, con más recursos que Arduino, pero con un modelo de programación que oculta determinadas capas de complejidad, facilita el planteamiento en el aula de problemas con soluciones divergentes [1], debido a que permite proyectos con distintas soluciones.

En este sentido, el empleo de un *framework* como el propuesto facilita el uso de la metodología de aprendizaje basada en proyectos. La razón es la disminución de los tiempos de desarrollo, lo que permite plantear proyectos más complejos y por lo tanto, más cercanos a la realidad. El análisis en clase de los distintos diseños propuestos por los alumnos puede ser usado para que estos discutan tanto su propio diseño como el de sus compañeros, permitiendo establecer procesos de auto y coevaluación [2]. Todo ello hace posible al profesor identificar con más claridad las habilidades alcanzadas por el alumno en los niveles de análisis, síntesis y evaluación establecidos por la taxonomía de Bloom.

6. Conclusiones

Este trabajo propone una modelo de programación basado en Arduino para Raspberry Pi, con el objeto de facilitar la programación de sistemas empotrados enmarcados en los paradigmas CPS e IoT. Este tipo de sistemas requieren multitarea y comunicación mediante TCP/IP. Dicho modelo, junto con su *framework*, ha empezado a utilizarse en una asignatura orientada a proyectos impartida a alumnos con carencias en la programación de aplicaciones con dichos requisitos. Los primeros resultados indican que los alumnos ven simplificado el desarrollo de sus proyectos, por lo que estos pueden tener un nivel mayor de complejidad con soluciones divergentes, facilitando así el empleo de la mencionada metodología docente.

Referencias

[1] Susan A. Ambrose, Michael W. Bridges, Michele DiPietro, Marsha C. Lovett, Marie K. Norman y Richard E. Mayer. *How Learning Works: Seven Research-Based Principles for Smart Teaching.* Jossey-Bass higher and Adult Education Series, mayo 2010.

[2] Thomas A. Angelo y K. Patricia Cross. *Classroom assessment techniques: a handbook for college teachers.* Jossey-Bass higher and adult Education Series, marzo 1993.

[3] Sergio Barrachina Mir, Germán Fabregat Llueca y José Vicente Martí Avilés. Utilizando Arduino Due en la docencia de la entrada/salida. *Actas de las XV Jornadas de Enseñanza Universitaria de Informática, Jenui 2015*, páginas 58 – 65, Andorra La Vella, julio 2015. Ponencia.

[4] Rebecca F. Bruce, J. Dean Brock y Susan L. Reiser. Make space for the Pi. *SoutheastCon Conference*, abril 2015.

[5] Pasquale Buonocunto, Alessandro Biondi y Pietro Lorefice. Real-time multitasking in Arduino. *9th IEEE International Symposium on Industrial Embedded Systems*, junio 2014.

[6] Alan Burns y Andy Wellings, *Real-Time Systems and Programming Languages Ada 95, Real-Time Java and Real-Time POSIX*. Cuarta edición, Addison Wesley Longmain, 2009.

[7] Carlos Catalán, Alfonso Blesa y Félix Serna. Industria 4.0 en el Grado de Ingeniería Electrónica y Automática. *Actas de las XV Jornadas de Enseñanza Universitaria de Informática, Jenui 2015*, páginas 327 – 332, Andorra La Vella, julio 2015. Ponencia.

[8] Zhuoqun Cheng, Ye Li y Richard West. Qduino: A Multithreaded Arduino System for Embedded Computing. *IEEE Real-Time Systems Symposium*, diciembre 2015.

[9] Jayavardhana Gubbia, Rajkumar Buyyab, Slaven Marusica y Marimuthu Palaniswamia. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Generation Computer Systems*, 29:1645–1660, septiembre 2013.

[10] Peter Jamieson y Jeff Herdtner. More missing the Boat - Arduino, Raspberry Pi, and small prototyping boards and engineering education needs them. *IEEE Frontiers in Education Conference*, octubre 2015.

[11] Ragunathan Rajkumar, Insup Lee, Lui Sha y John Stankovic. Cyber-physical Systems: The Next Computing Revolution. *47th ACM/IEEE Design Automation Conference*, junio 2010.