[Analógico • Digital • Microcontroladores & Embebido • Audio • Test & Medida]

Noviembre 2011 N.º 377 6,50 €

elektor

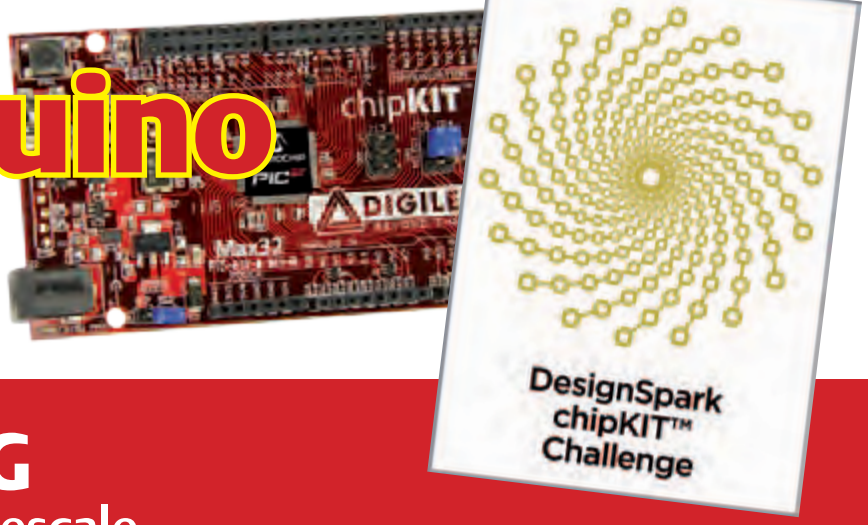
¿Cuánto estas expuesto? ¡Compruébalo!

Medidor de Radiación Mejorado

Para radiación alfa, beta y gamma

Super Arduino

Primeros pasos con el chipKIT™ Max21



Interfaz OnCE/JTAG

Programa y depura DSPs Freescale

Sencillo Detector de Murciélagos

Barato, sensible y fácil de construir

ISSN 0211-397X

00377

9 770211 397008

Gran expectación

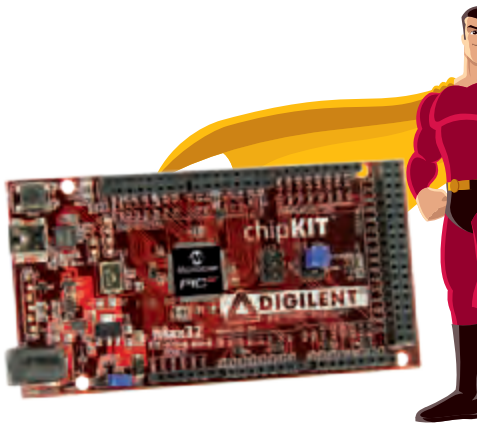
Algunos lo califican de “decepcionante”, otros de “problemática para adaptarse a los métodos de marketing activo”, pero yo lo encuentro fascinante y una constante fuente de inspiración y entretenimiento: proyectos y artículos que comienzan como algo insignificante, luego despegan de forma llamativa llegando eventualmente al estatus de gran éxito, aparentemente “autopropulsados”. El *Hexadoku, el E-Weekly, La Máquina del Caos, Pico C, Retrónica y el ElektorBus* son buenos ejemplos de cómo los lectores de Elektor determinan la proporción de éxito en lugar de nuestros redactores y editores.

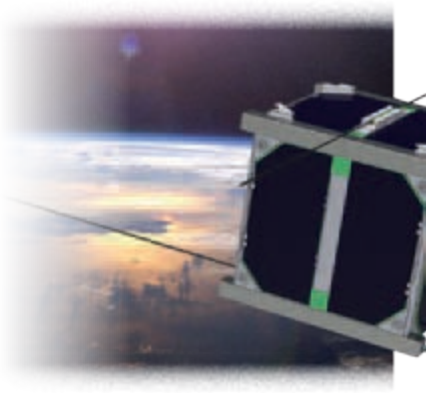
Hace ocho entregas, el *ElektorBus* no era más que una vaga idea mencionada de soslayo en una entrega de Labcenter. Desde entonces ha cobrado impulso y gracias a las muchas aportaciones de pensamiento activo estamos ahora en esta etapa describiendo un centro de control ElektorBus que se ejecuta sobre un Smartphone. Lo mismo con Retrónica, la sección creada por mi colega Jan, que empezó como una “broma”, pero que seis años después ha reunido un nutrido grupo de fieles seguidores – y ¡colaboradores!

Tengo varios ‘sensores’ disponibles para calibrar el éxito de los proyectos publicados. Los informes de venta de la revista y las placas son los indicadores evidentes que te vienen a la cabeza, pero como editor encuentro en las estadísticas semanales de la web un medio más rápido y más actualizado, también están las llamadas de teléfono y los emails en respuesta a un determinado artículo. En el buen espíritu de la ingeniería encuentro todas las formas de recibir información útiles y gratificantes.

Empezamos de nuevo un Desafío mundial, esta vez traído de forma conjunta por RS Components, Elektor y Circuit Cellar. Se trata de diseñar un proyecto basado en el hardware y software Digilent chipKIT™ Max32 utilizando la herramientas de diseño DesignSpark PCB de RS, donde idealmente los tres culminan en una placa “enchufable” que ayuda a ahorrar energía o reducir el consumo energético de cualquier forma que puedas imaginar. Tras el lanzamiento oficial del Desafío el 26 de Noviembre en el evento Elektor Live!, se distribuirá a los participantes una cantidad limitada de placas chipKIT™. En la página 14, Clemens Valens informa de sus primeras impresiones con el hardware y los componentes software chipKIT™ y puedes encontrar su exposición útil para estar en la primera línea de la parrilla de salida cuando el empiece el Desafío. ¡Espero también tu aportación!

¡Disfruta de esta edición!
Eduardo Corral, Editor





SUMARIO

Volumen 32
Noviembre 2011
nº 377

14 Super Arduino

La placa chipKIT™ Max32 ofrece una potencia de cálculo de 32 bits y 80 pines de entrada/salida mientras sigue siendo compatible con el entorno Arduino. Este es el componente hardware del excitante desafío de diseño que te traen RS Components, Circuit Cellar y Elektor. En este artículo encontrarás unos cuantos consejos que debería darte una ventaja para empezar a trabajar con el hardware y el software que vas a presentar para este desafío.

20 Robusta: un satélite realizado por estudiantes

Francia es conocida por su importante participación en el programa espacial europeo y por su industria espacial de primer orden, aunque menos por la formación de sus élites en este dominio. Por ello, el Centro Nacional de Estudios Espaciales (CNES) ha lanzado Expresso: la primera llamada para proyectos destinados a la enseñanza superior, en 2006. La Universidad de Montpellier 2 ha respondido con la presentación de su proyecto Robusta, un pico-satélite de tipo cubesat, que conlleva una experiencia científica útil a la comunidad espacial.

36 Sencillo detector de murciélagos

Muchas variedades de murciélagos producen sonidos en el rango de frecuencias de los 40 kHz, que está dentro del rango operativo de la mayoría de los transductores de ultrasonidos estándar. Si amplificas las señales recogidas por uno de esos transductores y las pasas por un divisor de frecuencias, la salida estará en tu rango audible de frecuencias.

42 Medidor de radiación mejorado

El instrumento básico que describimos en este artículo puede utilizarse con diferentes sensores para la medición de las radiaciones gamma y alfa. Es particularmente adecuado para medidas a largo plazo y para examinar muestras radiactivas débiles. El fotodiodo tiene un área sensible menor que un tubo Geiger-Müller así que tiene una tasa de recuento menor, lo que a su vez significa que es más fácil detectar la radicación de una pequeña muestra.

Super Arduino

Comenzando con el chipKIT Max32

Clemens Valens (Elektor Francia)

Si eres de los lectores interesados en microcontroladores, probablemente hayas oído hablar de Arduino y que, incluso, hayas trabajado con él. Si lo has hecho, también es posible que hayas rondado en los límites de esta encantadora plataforma de 8 bits, llegando a desear que tenga más potencia de cálculo, memoria e incluso líneas de E/S. Con la gran cantidad de placas de microcontroladores existentes, esto se puede conseguir fácilmente, pero con el coste de estar aprendiendo nuevas herramientas todo el tiempo. Esto ha sido así hasta hace poco, cuando la casa Digilent introdujo su solución para aquellos que deseaban más potencia sin tener que cambiar de herramientas. Su placa chipKIT Max32 ofrece una potencia de cálculo de 32 bits y alrededor de 80 terminales de E/S, al tiempo que sigue siendo compatible con el entorno Arduino.

Con toda seguridad, ya ha habido intentos previos de hacer las placas Arduino compatibles con 32 bits, sin embargo, hasta donde yo sé, estos intentos sólo han alcanzado un cierto factor de compatibilidad con Arduino y nunca una compatibilidad a nivel de las herramientas. Algunas de estas placas están soportadas por librerías de programas que ofrecen una funcionalidad y sintaxis similar a la de Arduino. Por su parte, Digilent ha llevado la compatibilidad con Arduino un paso más allá, integrando el compilador, el "linkador" y el programador para el procesador PIC32 usado en su placa chipKIT sobre el verdadero entorno de desarrollo integrado (IDE de ahora en adelante), Arduino 00222. Desde el punto de vista de un IDE Arduino, los chipKITs son sólo objetivos, cuidadosamente listados a lo largo de las placas Arduino clásicas de 8 bits. Diligent quiso incluso ir un poco más allá y llegar a crear un sitio web con una URL acabada en .cc [1], al igual que lo hace Arduino. Con el mismo espíritu de Arduino, la placa chipKIT comprende un código fuente y un circuito completamente abierto, lo que significa que los ficheros de diseño del circuito (esquemas eléctricos en Eagle y ficheros PCB), están disponibles para su descarga gratuita, así como los programas, todos ellos de código abierto. A diferencia de Arduino, la PCB chipKIT es una placa de cuatro capas, por lo que poca gente se atreverá a intentar fabricarla por sí mismo.

Hay dos modelos de chipKIT: el Uno32 y el Max32. Mecánicamente, el Uno32 es compatible con el Arduino Uno clásico y el

Max32 es compatible con el Arduino Mega, el cual es la versión más completa del Arduino estándar. Señalar que la casa Digilent ha hecho sus placas rectangulares para corregir la forma curiosa de borde corto de las placas Arduino de manera que son ligeramente más largas. Creo que esto no debe ser ningún problema para nadie.
El resto del artículo se centra en el Max32, así que, ¡vamos a ello!

La placa

La placa Max32 viene como una PCB roja de 4 capas, en una pequeña caja roja y blanca sin nada más, como por ejemplo, sin un cable USB, sin documentación, sólo con una URL [2]. Para aquellos que no conocen de memoria las dimensiones de la Arduino Mega, la placa mide 10,2 x 5,4 cm. Al igual que sucede en la Arduino Mega, tres bordes de la placa están alineados con los conectores, excepto el de los conectores para los terminales digitales del 0 al 13 (en idioma Arduino), que son modelos de doble fila en la placa Max32. Encontraremos los terminales digitales del 70 al 85 en los terminales extra. En el lado del conector de alimentación y USB, hay espacio suficiente para un conector de programación ICSP de Microchip. Este conector tiene el contorno especial tipo "Sparkfun" (escalonado) en la colocación de sus terminales, permitiendo que un conector de tipo "pinheader" realicen los contactos adecuados incluso si no están soldados.
La alimentación de la placa se puede hacer a través del conector USB o utilizando el conector "jack" de alimentación, que nos permitirá una tensión de entrada de hasta 15 Vdc. El puente JP1 nos permite saltar el regulador de 5 V, por lo que deberemos tener especial cuidado con lo que hacemos ya que podemos quemar el circuito integrado. Cuando damos alimentación a una placa virgen deberemos observar el molesto encendido del diodo LED rojo, colocado al lado del conector de alimentación, indicando que la línea de 3,3 V (3V3) proporciona alimentación mientras el diodo LED verde (LD4) parpadea a una frecuencia de 3 Hz.
El Max32 es poco más que una placa soporte (BoB, del inglés "breakout board") para el procesador PIC32 de 100 terminales, montado cerca del centro de la placa, con una fuente de alimentación y un puerto serie USB en el lateral. Este PIC32 es un microcontrolador de 32 bits de Microchip, que es muy similar a los Cortex-M3 de ARM (ver apartado correspondiente). La placa soporta el dispositivo PIC32 más grande disponible hasta el momento, el PIC32MX795F512L. Este circuito integrado está contenido en un encapsulado de 100 terminales y dispone de 512 KB de memoria flash, 128 KB de memoria RAM y trabaja con un reloj de 80 MHz. Dispone de USB-OTG (on-the-go), una MAC Ethernet y dos controladores CAN.
Después de esta corta introducción del circuito, vamos a echar una ojeada a los programas.

Introducción al PIC32

Cuando preguntamos por los microcontroladores de 32 bits, la mayoría de la gente menciona en primer lugar a ARM, otros piensan en aquellas casas que implementan el núcleo ARM, como Atmel, ST o NXP y, sólo unos pocos piensan en Microchip. Aunque es verdad que muchos teléfonos móviles confían en la tecnología ARM, una gran cantidad de otros consumidores de dispositivos electrónicos, como cámaras digitales e impresoras, contienen procesadores basados en MIPS. Hasta el momento no he hecho ninguna comprobación seria sobre esto, pero se dice que hay bastantes más procesadores MIPS de 32 bits en el mercado que de los procesadores ARM. Así pues, el tener alguna experiencia en procesadores MIPS es algo bueno para cualquier aficionado serio a los microcontroladores y el PIC32 es una excelente plataforma con la que comenzar.

Existe en la actualidad cinco familias: 3xx, 4xx, 5xx, 6xx y 7xx. La 3xx y la 4xx son consideradas de propósito general, mientras que las otras tres tienen más periféricos como CAN o Ethernet y pueden tener más memoria RAM. Los dispositivos están basados en un núcleo MK4 MIPS de 32 bits, con una transmisión de datos segmentada ("pipeline") de cinco etapas y capaz de soportar frecuencias a reloj de hasta 80 MHz. El fabricante asegura unas prestaciones máximas de 1,56 DMIPS/MHz (Dhrystone 2.1), un valor ligeramente mejor que el de un ARM Cortex-M3, que puede proporcionar hasta 1,25 DMIPS/MHz, según ARM.

Todas las familias tienen hasta 512 KB de memoria flash más 12 KB de memoria de arranque y hasta 32 KB de memoria RAM para los dispositivos 3xx/4xx o hasta 128 KB para los de las familias 5xx/6xx/7xx. Todos trabajan con todos los periféricos que podamos pensar para este tipo de microcontroladores (puertos serie, PWM, ADC, etc.) pero también disponen de múltiples canales de acceso directo memoria (DMA).

Las familias se suministran en dos «tamaños»: 64 terminales (sufijo H) o 100 terminales (sufijo L). Señalar que un encapsulado 121 XBGA contiene un dispositivo de 100 terminales. Los PIC32 son compatibles a nivel de terminales con algunos dispositivos PIC24 y dsPIC, por lo que pueden integrar una gran cantidad de microcontroladores de Microchip y de herramientas de desarrollo (MPLAB). En las páginas web existentes podemos encontrar una gran cantidad de librerías de programa a la vez que existen una gran cantidad de páginas web dedicadas a proyectos que comparten experiencias con PIC32 (www.mypic32.com). Las hojas de características y otro tipo de documentación también están accesible a través de la URL www.microchip.com/pic32.

Familia	USB OTG	CAN	Ethernet	RAM
3xx	–	–	–	Up to 32 KB
4xx	1	–	–	Up to 32 KB
5xx	1	1	–	Up to 128 KB
6xx	1	–	1	Up to 128 KB
7xx	1	2 (1 for 764)	1	Up to 128 KB

Figura 1: El IDE Max32 mostrando un `#define` para eliminar el código específico AVR.

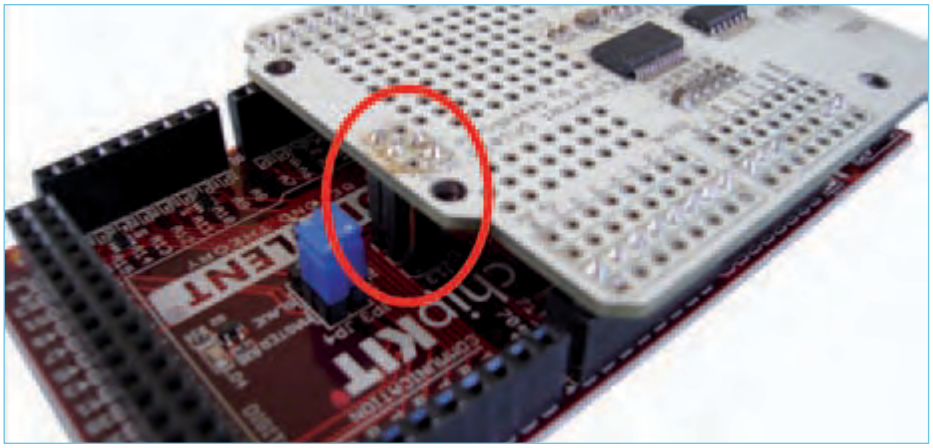


Figura 2: Una Mirada más cercana de la conexión SPI entre el “etherShield” y el Max32.

El IDE

Como ya se ha mencionado, la programación de la placa se ha hecho utilizando un IDE Arduino modificado que podemos descargar como archivo gratuito de [2]. La instalación del fichero de 128 MB es bastante sencilla, ya que sólo tenemos que descomprimirlo en un directorio adecuado, en el disco adecuado de nuestro ordenador. El programa descomprimido ocupa unos 480 MB de espacio de disco. Para iniciar el IDE tenemos que lanzar al ejecutable llamado **mpide.exe**, que se encuentra en el raíz del directorio IDE. El IDE es una “plataforma cruzada” (‘cross-platform’) que trabaja sin problemas sobre Windows, Linux y MacOS, aunque debemos tener instalado primero Java.

Al estar basado el Max32 en el IDE de Arduino, la gente de Digilent se ahorró el tener que escribir una gran cantidad de documentación. En su lugar, todo lo que necesitamos saber para instalar y comenzar a trabajar con Arduino se explica en detalle en la página web de Arduino [3]. También podemos ir a dicha página con nuestras preguntas sobre la sintaxis del lenguaje de programación.

En el momento de escribir este artículo, la versión del IDE era la 0022 (mpide-0022-chipkit-win-20110619 para ser precisos), la misma que la actual para el IDE Arduino oficial. Según Digilent, este IDE es idéntico al IDE Arduino official excepto en que ha sido ampliado con un compilador/linkador PIC32 y con librerías, de aquí que pueda ser utilizado para programar también las placas de 8 bits de Arduino. Por supuesto, lo he probado sólo para descubrir que mi clon Arduino, un Seeeduino v1.1, no fue reconocido: “Invalid device signature” («firma dispositivo no válida»). Esta placa funciona perfectamente con el IDE Arduino 0022 oficial.

Después de instalar el IDE y conectar el Max32 con el ordenador, podemos utilizar las herramientas para compilar uno de los ejemplos de muestra incluidos en el IDE y volcarlo sobre la placa. No debemos olvidar seleccionar la placa Max32 en el menú Tools -> Board y seleccionar el puerto serie adecuado correctamente (Tools -> Serial Port). Una vez hecho esto, el ejemplo BlinkWithoutDelay (File -> Examples -> Digital) debiera funcionar sin modificaciones con tan sólo pulsar sobre el botón *Upload* para ver el diodo LED verde LD4 que empieza a parpadear a una velocidad de 0,5 Hz.

Si hemos conseguido llegar hasta aquí sin problemas, tenemos configurado todo para desarrollar aplicaciones reales para el Max32. Así pues, continuad leyendo ya que aún habrá cosas que queremos saber...

Montando un shield

Hacer parpadear un diodo LED es bonito pero no muy satisfactorio, este es el motivo por el que me he aventurado a probar mi placa de extensión (shield) Ethernet Arduino en el Max32 (un shield es una placa de ampliación para una placa Arduino). La extensión está basada en el controlador independiente de Ethernet ENC28J60 de Microchip, con una interfaz SPI incorporado. Ya lo sé, el PIC32 ya incorporada una MAC Ethernet, pero no disponía de una ampliación manejable con un PHY Ethernet y un conector RJ45. Digilent propone esto como una ampliación (que ofrece otras cosas detrás), pero no tenía tiempo de pedir una y esperar a que llegase. Además, ahora ya he tenido una buena oportunidad para ver también como era la compatibilidad entre Arduino y el Max32. Como veremos más adelante, no es completa...

Mi vieja ampliación de Ethernet (a partir de ahora la llamaré ampliación-Ethernet o “etherShield”, para distinguirla de la ampliación Ethernet oficial de Arduino basada en el W5100), esta soportada por una librería y algunos ejemplos. Esta ampliación y su código funciona bastante bien sobre mi «Seeeduino». Por lo tanto, mi primer paso fue el de instalar esta librería en el IDE del Max32 (en el directorio *libraries*\) y ver si compilaba adecuadamente. La respuesta fue, como bien podemos esperar, NO. La razón no estaba en el propio código sino en el hecho de que el compilador, aparentemente, era incapaz de encontrar algo que compilar. Según la página web de Digilent, donde se explican algunas cosas a tener en cuenta en la exportación del código Arduino, las librerías se supone que se manejan de la misma manera a como lo hace Arduino, pero, claramente, ese no es mi caso. Cuando coloqué la librería en el directorio *hardware**pic32**libraries*\ (donde encontramos los mismos elementos que en el directorio *libraries*\), el compilador encontró el código, pero produjo muchos errores y avisos diciendo que el código contenía código específico de AVR (las placas Arduino están basadas en los procesadores AVR de Atmel). ¡Estupendo! Ahora ya tenía por dónde empezar.

Lo primero que tenemos que hacer cuando exportamos las librerías Arduino es deshacernos de las referencias a la memoria de programa. Con un AVR son necesarias algunas directivas especiales para el

compilador para que pueda acceder a las constantes (strings, tablas) localizadas en el espacio de programa. Esto no es necesario para el PIC32 y estas directivas tienen que ser eliminadas. Para mantener nuestro código Arduino compatible sería mejor definir estos parámetros (`#define`) en otro lugar. Así, podemos usar la macro `_BOARD_MEGA_` (ver **Figura 1**), definida por el IDE Max32, para hacer esto (aclaración: es posible que los lectores esperasen algo como `_BOARD_MAX32_`). Debemos hacer lo mismo para cualquier directiva `#include` específica de AVR.

Es posible que esto no sea suficiente (en mi caso no lo fue) ya que la librería también puede hacer referencia a registros del AVR que el PIC32 no tiene. El controlador SPI para el circuito integrado Ethernet ENC28J60, se comportó así, probablemente porque es relativamente viejo y la librería SPI, que ahora forma parte del IDE Arduino, sencillamente no existía cuando fue escrita en su momento (no aparece antes de la versión 0019 septiembre 2010). Así pues, modifiqué la vieja librería etherShield para utilizar en su lugar la nueva librería SPI Arduino y probé esto primero en IDE real de Arduino antes de intentarlo en el IDE del Max32.

Esto introdujo nuevos problemas ya que la librería SPI hace referencia a funciones de terminales que al compilador Max32 no le gustan. Al final, el problema estaba causado por mi librería que estaba escrita en C y compilada como tal, mientras que las funciones de los terminales y la librería SPI habían sido escritas en C++. Así pues, de vuelta de nuevo sobre el IDE de Arduino, tuve que exportar la librería etherShield completa a C++ y probarla. Esto no fue particularmente difícil de hacer, pero tenemos que vigilar directivas del compilador como `#extern «C» { … }`, ocultas en lugares inesperados. Después de las últimas modificaciones mi librería etherShield se compiló sin errores en el IDE del Max32 *y* para la placa Max32. Pero, ¿funcionó?

¡No, por supuesto que no! Realmente esto no fue una sorpresa ya que ya había visto problemas en los mensajes sobre SPI, en la página web de Digilent, pero el optimismo se mantenía en mí.

El principal problema es la incompatibilidad de los terminales de E/S del Max32 con los terminales de E/S del AVR. Los terminales digitales de Arduino del 10 al 13 tienen la capacidad de trabajar con SPI, mientras que los terminales 10 y 11 lo pueden hacer también con PWM. La razón para esta combinación es sencillamente el modo en que los terminales

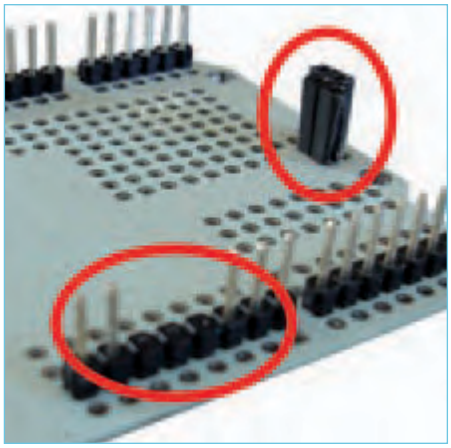


Figura 3: Las modificaciones que he hecho en el “etherShield”.

Figura 4: El resultado de un puerto conseguido: ahora ya puedo conectar un diminuto servidor web que corra sobre el Max32.

de E/S del AVR han sido sacados por Atmel. El PIC32 combina las funciones de un modo diferente en sus terminales de E/S y no existe una equivalencia exacta con los terminales del AVR. Diligent ha elegido dar prioridad a las funciones PWM, tal y como han sido usadas en las funciones analogWrite de Arduino, lo que significa que sólo podría conectarse parte del puerto 2 SPI a estos terminales. Sin embargo, han encontrado un modo de conectar el puerto 1 SPI de un modo compatible con Arduino, usando el conector ICSP Arduino (ver **Figura 2**), que está conectado a las mismas señales que los terminales 10 a 13. Nunca he considerado este conector como parte de una ampliación compatible pero he tenido suerte, ya que Seeedstudio, el fabricante de mi etherShield, ha colocado un conector tipo “pinheader” ICSP en la ampliación, en el lugar adecuado. Su sustitución con un conector hembra en la cara de soldadura sólo me llevó unos pocos minutos y restableció la compatibilidad SPI con el Max32. Para evitar conflictos entre los terminales 11, 12 y 13 (MOSI, MISO y SCK), simplemente suprimí estos terminales de mi ampliación (ver **Figura 3**). Ahora ya debería funcionar, ¿no es así?

Error. En este punto encendí mi osciloscopio ya que sospechaba problemas de compatibilidad entre el protocolo SPI producido por el PIC32 y el aceptado por el ENC28J60. Lo más notable fue la velocidad de reloj que me perjudicaba ligeramente. El osciloscopio me demostró que estaba en lo cierto ya que donde el Arduino intentaba alcanzar una velocidad de reloj de unos 610 kHz, el PIC32 estaba volando en torno a los 20 MHz. De acuerdo a las hojas de características del ENC28J60 esto debería ser válido, pero haciendo pruebas más tarde, cuando todo funcionaba adecuadamente, me demostró que una frecuencia de 2,5 MHz era un valor mucho más realista. Por ahora, tan sólo he reducido la velocidad de reloj del PIC32 a un valor parecido al de Arduino: 625 kHz. Con este cambio la placa de ampliación aún no funcionaba pero podía sentir que estaba muy cerca.

Hoy día, incluso los osciloscopios digitales de presupuesto más bajo, pueden grabar las curvas de pantalla, incluyendo mi Atten ADS1022C, de 25 MHz y 240 € (portes incluidos), donde esta función tan útil me demostró que había un problema de polaridad/fase entre el reloj SPI y las líneas de datos. Comparando cuidadosamente las transiciones descubrí que la placa de ampliación necesitaba el modo 1 SPI en el Max32, mientras que se usaba el modo 0 en la placa Arduino. ¿El hacer esto significaba que la placa de ampliación iba a funcionar por fin? Así fue (ver **Figura 4**). ¡Uf!.

Puntualizaciones finales

Digilent ha hecho un trabajo decente en la exportación del PIC32 al IDE Arduino. Aunque no hayan alcanzado el 100% de compatibilidad, sí que han trabajado duro y han conseguido llegar hasta casi conseguirlo. Se suele suponer que una sencilla placa de ampliación Arduino, con unas sencillas librerías que respeten las reglas y el estilo del código de Arduino, serían fácilmente exportables, aunque podamos caer en algunos de los problemas en los que yo me he encontrado. Para las placas de exportación más complejas de AVR nos podemos encontrar con pequeñas sutilezas que serán mucho más difíciles de solventar y que pueden requerir unos sólidos conocimientos sobre el PIC32. Para hacernos la vida más fácil, intentemos utilizar las funciones disponibles en las librerías Arduino en lo máximo posible y dejar que Digilent haga el trabajo duro por nosotros. Digilent ha comenzado a mantener una lista de placas de ampliación «reconocidas que funcionan» por lo que recomendamos echar una ojeada antes de comenzar nuestro propio proyecto. Podemos esperar a las actualizaciones del IDE Max32 que corrigen algunos de los problemas mencionados en este artículo, por lo que debemos asegurarnos que siempre estamos utilizando la última versión.

La única cosa que aún no he llegado a tener completamente clara es el problema de dónde colocar nuestras propias librerías. Después de pensar mucho en ello y hacer algunas pruebas, he asumido que todos los ficheros que contienen código específico del PIC32, como los controladores de bajo nivel, deben estar colocados en el directorio *hardware\pic32\libraries*, incluidos los ficheros que necesitan estos ficheros. El resto de ficheros, incluyendo los ficheros ejemplo que se usan en la librería, deben estar colocados en el directorio *libraries* para asegurar que van a ser reconocidos por el IDE como ejemplos.

Los ficheros con el código fuente para las pruebas y los experimentos descritos en este artículo pueden ser descargados de [4].

(110661)

Enlaces en Internet y Recursos

[1] http://chipkit.cc
[2] www.digilentinc.com
[3] http://arduino.cc
[4] www.elektor.es/110661

Robusta: un satélite realizado por estudiantes

Unos pico-satélites para promover la educación espacial

Frederic Giamarchi (Francia)

Francia es conocida por su importante participación en el programa espacial europeo y por su industria espacial de primer orden, aunque menos por la formación de sus élites en este dominio. Por ello, el Centro Nacional de Estudios Espaciales (CNES) ha lanzado Expresso: la primera llamada para proyectos destinados a la enseñanza superior, en 2006. La Universidad de Montpellier 2 ha respondido con la presentación de su proyecto Robusta, un pico-satélite de tipo *cubesat,* que conlleva una experiencia científica útil a la comunidad espacial.

El satélite Robusta (***R**adiation **O**n **B**ipolar **U**niversity **S**atellite **T**est **A**pplication*, ver **Figura 1**) [1] será lanzado por la nueva lanzadera europea Vega, a principios de 2012. Transportará un experimento científico de la medida de la degradación de los componentes electrónicos. Se situará en una órbita elíptica de 340 km a 1450 km, con una inclinación de 71°. Durante toda la duración de su vuelo, transmitirá a la estación de tierra de estudio, localizada en el campus de la Universidad de Montpellier, los datos de las medidas de los componentes probados y los diversos parámetros de su estado. Sometido a diversas fuentes de radiación, viento solar, partículas que impactan de los centenares de radiaciones y rayos cósmicos, el satélite descenderá poco a poco y se desintegrará al cabo de dos años de vida, al entrar de nuevo en la atmósfera terrestre.

El satélite Robusta

Este satélite tiene una verdadera misión científica: medir la degradación de dos años de radiación ionizante de componentes electrónicos basados en transistores bipolares. Los componentes a probar

Los cubesat

Los satélites del tipo «cubesat» corresponden a un programa de enseñanza puesto en funcionamiento en el año 2000 por la Universidad Politécnica de California (CalPoly) [2], cuyo objetivo es el de ofrecer a los estudiantes una experiencia concreta así como conocimientos punteros ligados a la investigación y a la industria espacial. Un cubesat es un satélite cúbico de 10 cm de lado que pesa 1 kilo como máximo y dispone de una potencia máxima de 1 W. Está constituido por una carga útil, llamada *payload,* que corresponde al experimento enmarcado y de una plataforma que comprende las diversas placas electrónicas que permiten el control del experimento, la comunicación con la estación de tierra y la gestión de la energía. El conjunto representa el equivalente de un satélite convencional de tamaño muy pequeño, sometido a las mismas limitaciones que los grandes satélites, con choques térmicos, vibraciones extremas en el despegue, radiaciones, vacío en el espacio, etc.

elegidos son los comparadores de tensión LM139 y los amplificadores de tensión LM124, componentes utilizados a menudo a bordo de los satélites. Estas degradaciones se cuantifican por medio de medidas de corriente, de tensión, de temperatura y de la dosis recibida (ver **Figura 2**). Esta dosis corresponde a la cantidad de energía por unidad de masa. A continuación, los resultados serán comparados con los obtenidos por un método de prueba, en tierra, puesto en funcionamiento por investigadores de los laboratorios de la Universidad IES (Instituto de Electrónica del Sur) de Montpellier [3][4].

La duración de la misión en vuelo se ha fijado en dos años. Los datos serán medidos, como mínimo, cada 12 horas. Seguidamente, serán transmitidos a la estación de tierra del campus de Montpellier utilizando frecuencias y un protocolo de radioaficionados. La transmisión se hará en radiodifusión ("*broadcast*"), es decir, en continuo, cada minuto, independientemente de que el satélite esté en una ventana de «visibilidad» de la estación de tierra o no.

Un punto crucial para el éxito de la misión es la gestión de la energía. La alimentación del satélite se hará por medio de baterías de Ión-Li, de la casa Saft, que se recargarán por células solares específicas espaciales, de triple unión, y con un rendimiento del 27%.

La estructura interna

La estructura mecánica, las dimensiones, el posicionamiento de las placas electrónicas y la problemática del sistema de lanzamiento, por medio de un "p-pod" (ver **Figura 3**), son realizadas por la sección GMP (Génie Mécanique et Productique, es decir, Ingeniería Mecánica y de Producción) y la GEII (Génie Électrique et Informatique Industrielle, o Ingeniería Eléctrica e Informática Industrial) del IUT de Nîmes. El sub-sistema de la placa de alimentación, la gestión de la energía de las baterías y de las células solares, quedan asegurados por la sección GEII del IUT de Nîmes. El sub-sistema de la placa del controlador, que gestiona las órdenes entre las placas y almacena los datos de medida, y las partes de programación de microcontroladores o receptores de las pruebas, son llevadas a cabo por las secciones del Polytech de Montpellier. El sub-sistema de la placa de experimentación, que comprende los componentes bajo prueba, el sensor de la dosis de radiación y los sensores de temperatura, ha sido concebido por estudiantes de licenciatura y de máster EEA de la facultad de ciencias. Los sub-sistemas de la placa de radiocomunicación y la estación de tierra, han sido elaborados más particularmente para el recorrido en hiper-frecuencias.

Los componentes y materiales utilizados en este proyecto son componentes comerciales no rugerizados, a excepción de algunos elementos como la batería y las células solares. Se ha implementado un procedimiento riguroso para asegurar la calidad de la radiación y reducir en lo máximo posible los riesgos unidos a su exposición a las radiaciones. Se considera que el proyecto será un éxito completo si se sobrepasa un año de funcionamiento.

La estructura mecánica

La estructura mecánica ha sido diseñada y fabricada en bloque para que todo forme una única pieza (ver **Figura 4**). Los diversos elementos del satélite, células solares, PCIs, tornillos, conectores, hilos, etc., han sido todos diseñados y dimensionados a medida que avanzaba el proyecto. Ha sido necesaria una constante interacción entre los diferentes equipos de las diferentes partes para actualizar los diversos elementos, a medida que se iban validando las distintas evoluciones y correcciones. La estructura está realizada en aluminio 6061 de densidad estable en el medio espacial.

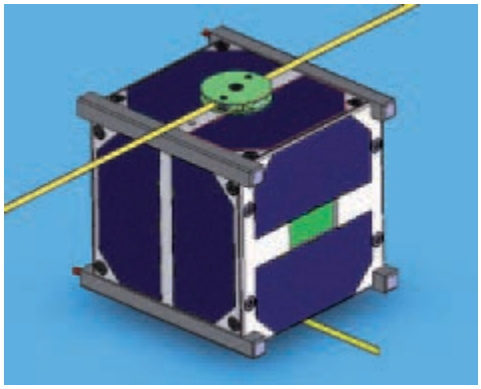


Figura 1. Modelo CAD del satélite Robusta. (fuente: RobustaCom)

La placa de potencia

La placa de potencia recargar la batería y distribuye las diversas tensiones necesarias de los diversos sub-sistemas. Aquí también se encuentra el sistema necesario para la activación del despliegue de

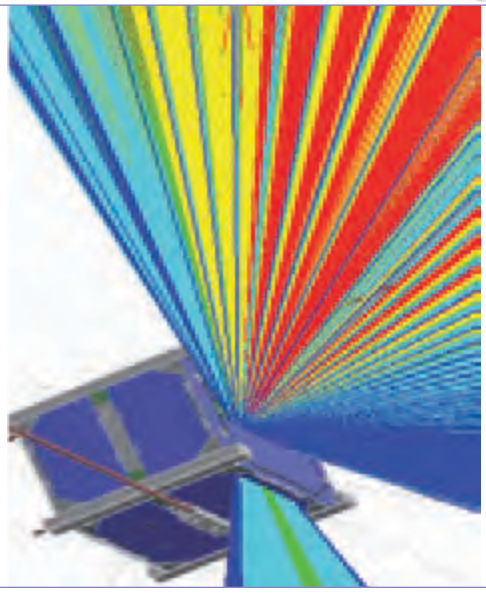


Figura 2. Ejemplo de modelización para el cálculo de la dosis absorbida por un componente de Robusta que utiliza el programa FASTRAD. (fuente: RobustaCom)

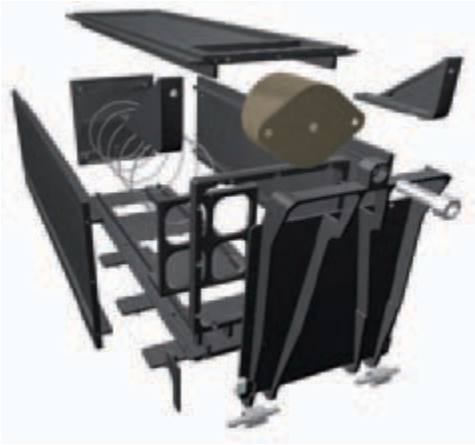


Figura 3. Modelo interconectado 3D de un “p-pod”. (fuente: CubeSat Project)

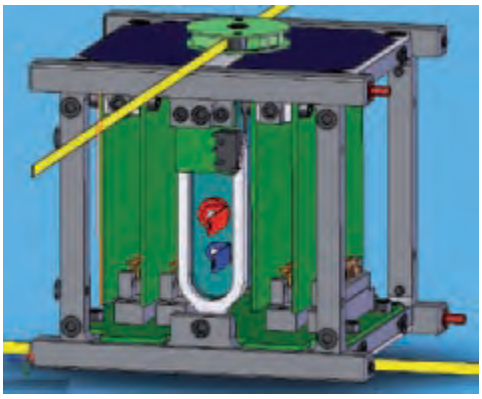


Figura 4. Modelo interconectado 3D del satélite. (fuente: RobustaCom)

las antenas, una vez que el satélite está en órbita. Las dimensiones de los componentes permiten disponer de un margen adecuado a la potencia necesaria. El sistema de carga de la batería tiene en cuenta la variación de la tensión de las células solares en función de la temperatura y de la degradación de éstas con el tiempo. Se proporcionan tres tensiones: 8 V, para la amplificación antes de la emisión de datos hacia la Tierra; 6 V, para los circuitos lógicos; y -5 V para los componentes bajo prueba. Las seis caras que albergan las células solares estarán sometidas a las radiaciones solares en un orden aleatorio, siguiendo la rotación del satélite. Se ha elegido medir la tensión y la corriente proporcionada por estas seis caras, con el fin de verificar la carga adecuada de las baterías y medir la rotación implícita del satélite. Los estudiantes han elegido un bus I²C para dialogar entre la placa de potencia y la placa del controlador.

La placa de experimentación

Para la placa de experimentación, su circuito electrónico ha debido ser diseñado y probado ampliamente. En efecto, cada circuito integrado bajo prueba (LM124 y LM139) comporta ocho elementos sobre los cuales es necesario medir sus corrientes, tensiones, temperaturas y dosis de radiación. Así pues, ha sido necesario encontrar una arquitectura, basada en conmutadores analógicos controlados por el microcontrolador, para multiplexar las medidas tomadas sobre los diferentes terminales de los componentes (ver **Figura 5**). Los estudiantes han tenido que elegir un bus que permitiese gestionar el gran número de direcciones asignadas a los interruptores y, de esta forma, descubrir los buses I²C y SPI. Además de la placa de potencia, las otras placas poseen en común un PIC18F4680, un interfaz CAN y un dispositivo *anti latchup* (protección de los microcontroladores contra los cortocircuitos generados por partículas ionizantes).

La placa controlador

Es el cerebro del satélite: su función es la de ordenar las tareas de las otras placas. Gestiona el diálogo con las otras placas y es responsable de la gestión de la energía disponible. Por ejemplo, prohíbe una comunicación con la estación de tierra cuando hay una prueba en curso, ya que estas dos acciones, si se ejecutan al mismo tiempo, consumirían demasiada energía.

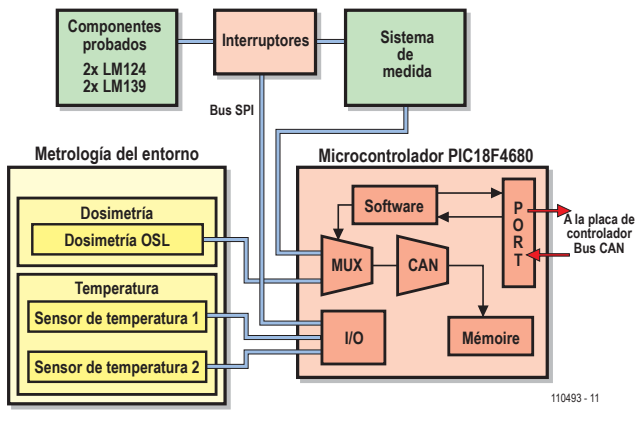


Figura 5. Diagrama de bloques de la placa experiencia. (fuente: RobustaCom)

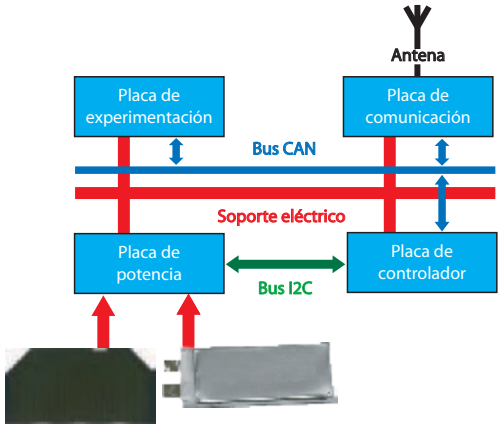


Figura 6. La interconexión de las cuatro placas con interfaz I²C y bus CAN.

100 % estudiante

Robusta, satélite y estación de tierra dedicada, han sido enteramente diseñados y realizados por estudiantes provenientes de diversas formaciones de la Universidad de Montpellier 2: IUT de Nîmes, escuela de ingenieros Polytech'Montpellier, licenciados y masters EEA (electrónica, electrotécnica, automática) de la Facultad de Ciencias. Este proyecto, en cooperación con el CNES, está financiado por otras grandes industrias del sector electrónico.

por encima de la media. El satélite debe estar pensado, realizado y verificado por los estudiantes, guiados por profesores expertos en el dominio asociado, siempre bajo el control del CNES. Igualmente, los estudiantes se encargan de la gestión del proyecto.

Este proyecto es una experiencia de naturaleza industrial que comparte su duración, su coste y su tecnología a la escala de los estudiantes. En cuanto al sistema, Robusta permite que los estudiantes

De este proyecto se podrán sacar varias grandes líneas: el estudio del sistema y la gestión asociada, la estructura mecánica, las pruebas medioambientales, los sub-sistemas relativos a diversos dominios del EEA: la gestión de la energía, la carga útil, la placa controladora, la placa de radiocomunicación, la estación de tierra asociada. Por último, falta añadir una parte de comunicación y de divulgación de bachillerato puedan desarrollar prototipos técnicos punteros y mejorar sus sentidos de la comunicación, todo ello descubriendo el mundo espacial. Deben implicarse desde la definición de la misión hasta la explotación de los datos de medida, pasando por todas las fases de diseño, aprovisionamiento de materiales, realización de prototipos y pruebas.

Durante las reuniones celebradas entre los distintos sub-sistemas, se ha decidido utilizar un bus CAN para la comunicación entre los diferentes sub-sistemas (ver **Figura 6**). Por otro lado, durante la concepción del prototipo, los estudiantes han podido desarrollar su propio protocolo de intercambio de datos. Como el número de mensajes a enviar a las otras placas era elevado, se ha decidido utilizar una red Pétri con el fin de no publicar mensajes, evitar bloqueos y, además, tener en cuenta las limitaciones impuestas.

La placa de radio

Para el sub-sistema de radiocomunicación, los estudiantes han comprendido el sentido de banda de frecuencias asignadas para una aplicación. Después de haber estudiado diversas arquitecturas de emisión-recepción, han optado por un sistema de dos frecuencias separadas: 435,325 MHz para la emisión hacia la estación de tierra y 145,95 MHz para la recepción de los telecomandos. La elección de los componentes y, principalmente, los amplificadores, se ha hecho directamente de acuerdo con los estudiantes del sub-sistema de la estación de tierra, en función del presupuesto del enlace. Sin embargo, además de una cultura necesaria sobre la problemática propia a las radiofrecuencias, los estudiantes han debido poner en marcha unos procesos de tratamiento de señal en función de la elección del tipo de modulación y demodulación. No se han olvidado de la simulación, principalmente para las antenas del satélite, que han sido simuladas enteramente con la ayuda de un programa profesional específico para las hiper-frecuencias: el CST Microwave Studio.

La estación de tierra

La estación de tierra es una parte integral e indispensable para el funcionamiento de una misión espacial. Cuando el satélite está en órbita, se convierte en el único interfaz de comunicación posible. Así, permite recibir todos los datos de experimentación y los parámetros de vuelo (telemetría), pero también enviar los telecomandos para modificar el protocolo experimental o el comportamiento del satélite (por ejemplo, gestionar la alimentación, activar o desactivar ciertas partes del satélite).
La estación de tierra de Robusta (ver **Figura 7**) está construida alrededor de equipos de radioaficionados. El elemento central de su arquitectura es un emisor-receptor que permite la modulación/demodulación de las señales AFSK, en la banda de los 430MHz para la teleme-

Expresso

El Centro Espacial de Toulouse (CST), que depende del CNES, propone a los estudiantes adquirir una experiencia concreta en el dominio de los sistemas orbitales. También es la ocasión de probar nuevas tecnologías y de realizar experimentos científicos para la comunidad espacial a mitad de coste. Para mantener el proyecto, el CNES ofrece medios financieros y pone a disposición un coordinador del proyecto y de expertos del CST para el análisis térmico, las células solares, las pruebas de vibración, las autorizaciones de uso de frecuencias, etc.

Figura 7. La estación de tierra de Robusta. (fuente: RobustaCom)

Figura 8. El cohete Véga. (fuente: ESA)

tría y en la banda de los 144 MHz, para los telecomandos. Dos antenas motorizadas establecen el enlace con el satélite. Un ordenador gestiona la motorización de estas antenas así como el envío de los telecomandos y la recepción de la telemetría. Todos los programas de la estación de tierra han sido desarrollados, bien de manera interna, o bien recuperados del mundo del «código abierto», comenzando por el sistema operativo del PC, que es Linux Ubuntu y que gestiona la estación de tierra. Esto permite la adaptación de los programas a nuestras necesidades específicas y una evolución en el tiempo sin limitaciones particulares.

La lanzadera Vega

Después de una petición de candidatos, se ha seleccionado el satélite Robusta, entre otros ocho, para su puesta en órbita tras el vuelo de calificación de la lanzadera Vega (ver **Figura 8**). El proyecto Vega debe permitir poner en órbita satélites de pequeño tamaño, de entre 300 y 2000 kilos, en órbitas bajas o polares. Esto será una primicia para esta lanzadera, que despegará de la base de Kourou a finales de 2011.
En total, serán expulsados de la lanzadera un total de nueve cubesat al mismo tiempo que la carga principal, un satélite científico llamado LARES System, así como el mini-satélite educativo ALMASat.

(110493)

Prueba de vibración del cubsat sobre un vibrador (fuente: Cnes)

Enlaces en Internet & referencias

[1] El proyecto Robusta: www.ies.univ-montp2.fr/robusta/

[2] Cubesat de California Polytechnic State University : http://polysat.calpoly.edu/

[3] J. Boch, *"Estimation of Low Dose Rate Degradation on Bipolar Linear Integrated Circuits Using Switching Experiments"*, IEEE Trans. Nuclear Science, vol. 52, p.2626-2621, diciembre. 2005.

[4] J. R. Vaille, F. Ravotti, P. Garcia, M. Glaser, S. Matias, K. Idri, J. Boch, E. Lorfevre, P. J. McNulty, F. Saigne, L. Dusseau, *"Online dosimetry based on optically stimulated luminescence materials"* IEEE Trans. on Nuclear Science, vol. 52, Issue 6, Diciembre 2005 pp. 2578 – 2582.

La aventura espacial continúa con Friends y Ristretto

Robusta, Expresso, Friends, Ristretto, no son términos relativos al café sino, más bien, los nombres de los proyectos de estudiantes para conquistar el espacio. Pequeños satélites de estudiantes, cada vez más grandes; esto es lo que nos propone la Universidad de Montpellier 2 en colaboración con el CNES, la ESA y la Universidad Baumann de Moscú.

Para hacer frente a estos nuevos retos, la Universidad de Montpellier 2 próximamente va a poner en funcionamiento un centro espacial universitario, bautizado SOLARIUM (Sistemas Orbitales Ligados a las Actividades de Investigación Interdisciplinaria de la Universidad de Montpellier 2) y subvencionado por la fundación Van Allen, primera en Francia en valorar los pequeños satélites.

Curso de audio DSP

Parte 5: Estructura de los programas DSP del curso

En el siguiente artículo describimos las estructuras de aplicación de los programas diseñados en el curso DSP. Hemos optado por hacer una sección específica para todas las aplicaciones dentro del mismo entorno y los programas individuales para el tratamiento de señales en un bucle de audio (audioloop). El objetivo es facilitar el acceso a la programación DSP y el uso de los programas descritos en el curso. El artículo concluye con algunas notas y consejos útiles sobre el ensamblador y la programación de DSPs.

Alexander Potchinkov (Alemania)

El software se divide en dos partes. La primera referente al *entorno de programa*, que es común para todas las aplicaciones. La segunda es un *audioloop* en el cual se procesa la señal, y que es única del proyecto del curso y permite al lector llevar a cabo otras aplicaciones individuales. Ambos conceptos se describen en el presente artículo. Para este curso las traspasaremos y adecuaremos a nuestra tarjeta DSP.

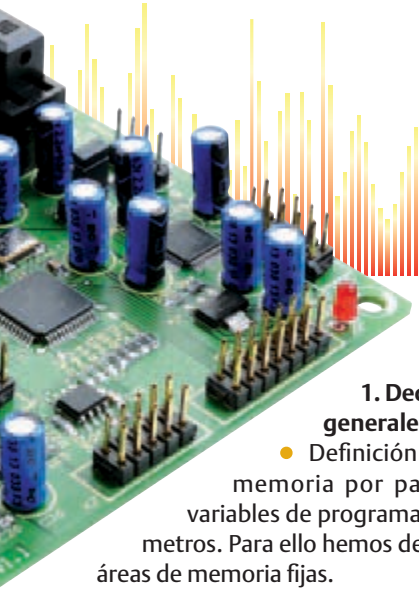
Tareas del entorno de programa

En el entorno podemos definir por nuestra cuenta varios puntos del DSP: reloj del procesador y sistema de interrupciones, controlar el stack de software y muchos más. Entre otros, el DSP con los componentes periféricos se conecta con la tarjeta DSP, y hace posible el intercambio de datos de audio y de control entre sí. La conexión con los componentes periféricos significa que en los registros del DSP se describirán explícitamente las configuraciones de los puertos DSP. Podríamos decir que en el entorno de programa se configuran las funciones del hardware, lo cual posibilita que trabajemos con él. En nuestro caso esto abarca cinco componentes periféricos, siendo normalmente el ADC, DAC, SRC, la SEEPROM y el puerto SPI, los cuales son para nosotros como los "planetas alrededor del sol DSP". La **figura 1** muestra dicho "sistema solar". Las líneas continuas representan la transmisión de datos y las de puntos la transferencia de señales de control, que sirven tanto para programar como para transmitir información sobre el estado del funcionamiento y los componentes periféricos.

- Para el **ADC** se disponen las señales del reloj, con las cuales se definen la frecuencia de funcionamiento y la planificación de las transferencias de datos. El ADC es capaz de detectar por su cuenta la relación entre la frecuencia principal y la de funcionamiento del DSP y por ello no hace falta programarlo. En nuestro caso la relación es de 512.
- Para el **DAC** también hay que poner a disposición las señales del reloj. Al igual que el ADC, el DAC también es capaz de detectar la relación entre frecuencias y configurarse debidamente.
- El **SRC** no funciona tan "independiente" como el ADC y DAC, sino que hemos de programarlo. Los puertos I²S y de audio digital del SRC se configuran mediante las series de bytes de la tercera parte del curso. La programación del DSP se hace vía SPI y forma parte del propio entorno. Esto requiere configurar el SPI de modo que el protocolo de transferencia se corresponda con el SRC. Finalmente se reinicia el SRC desde el DSP y se transfieren 54 bytes al SRC.
- La **SEEPROM** puede utilizarse como memoria no volátil. Si queremos que la tarjeta DSP funcione con una determinada aplicación fija sin hacer uso del debugger, podemos iniciar el DSP desde la SEEPROM. El DSP contiene un programa bootloader en una de las partes de su ROM.
- Mediante el **puerto SPI** pueden transmitirse datos a componentes externos fuera de la tarjeta DSP. En todos los medidores digitales de nuestro curso utilizaremos el puerto SPI para controlar los displays de barras LED. Después también podemos añadir al puerto SPI por ejemplo un microcontrolador con teclado, un potenciómetro digital (codificador rotativo) y un LCD, y así poder cambiar los parámetros de nuestro programa DSP, por ejemplo ajustar las frecuencias de oscilación de nuestro generador sinusoidal.

Diseño del entorno

El entorno de programa consta de cuatro partes:



1. Declaraciones generales

- Definición del uso de memoria por parte de las variables de programa y los parámetros. Para ello hemos determinado áreas de memoria fijas.
- Asignación de las constantes requeridas por el programa.
- Entrada de datos en el vector de interrupciones, que se encuentra en el archivo `ivt.asm`.

2. Configuración del DSP

Los ajustes del DSP (la configuración del núcleo del procesador y los puertos periféricos) se lleva a cabo escribiendo registros de 24 bits, utilizando direcciones que van desde la $FFFF80 a la $FFFFFF. Este espacio incluye 128 direcciones, de las cuales utilizamos 18, y según el fabricante del procesador se conoce como *Internal I/O Memory Map*. En nuestro DSP56374 con encapsulado de 52 pines nos servimos de la X-RAM. La escritura de los registros se lleva a cabo mediante la instrucción `movep` (Move Peripheral Data), que permite escribir en los que han sido seleccionados sin tener que utilizar el registro de procesador, como ocurriría con la instrucción de `move` "normal". Como ejemplo, la instrucción `movep #$D17D00,x:RCR`, con la cual se activa el RCR (Receive Clock Register). Este registro especifica las características de los módulos receptores de los puertos de audio. `RCR` es sólo una abreviatura útil (igual que las reglas nemotécnicas en lenguaje ensamblador) tras la cual se esconde la dirección $FFFFBF, que difícilmente seremos capaces de recordar. Por esta razón, en cada programa utilizamos el archivo de ayuda `mioequ.asm`, en el cual, entre otras cosas, se listan las direcciones de todos los registros de entrada y salida con abreviaturas útiles, facilitando el trabajo al programador enormemente. Este archivo se enlaza con la rutina Include en ensamblador.

- El programa empieza con el contador de programa $000000, ocupado por el reset del DSP. La primera instrucción es un salto a la dirección de programa $100, que incluye la tabla de vectores de interrupción y otras rutinas internas. Aquí es donde comienza el verdadero programa.
- Se bloquean todas las interrupciones para las siguientes configuraciones del procesador. Así también evitamos los "accidentes", mientras el procesador se inicializa con sus registros del núcleo y los periféricos.
- Se resetea el puntero de stack del hardware (sp) y se inicia el stack del software en la X-RAM a partir de la dirección $40, con las direcciones ascendentes.

Las configuraciones del DSP y sus periféricos son bastante completas, ya que el DSP está pensado para ser muy flexible y universal. No obviaremos decir que el ajuste de un DSP sólo puede hacerse revisando exhaustivamente el manual, lo cual puede suponer toda una tortura para el principiante. Por suerte, el procesador se ha diseñado "siguiendo una lógica" y el manual de Freescale está bien redactado. Una detallada descripción paso a paso nos pondrá en situación sin mayores problemas. En adelante sólo damos las indicaciones más importantes:

- **Ajustes del PLL para el reloj del procesador**. El reloj del procesador está configurado para funcionar a 147,456 MHz, lo cual equivale a seis veces la frecuencia del master de audio y a 3072 veces la de muestreo, de 48 kHz. Con los factores del preescalado y el multiplicador, el hemos de prestar atención al rango de frecuencia VCO permisible de 300 a 600 MHz (ponemos el VCO a una frecuencia de 589,824 MHz). Podemos encontrar en las tablas las indicaciones para una frecuencia de audio más comunes. Para frecuencias distintas tendremos que hacer los cálculos por nosotros mismos. Damos por hecho que la tarjeta está equipada con un oscilador de cuarzo a 24,576 MHz. El PLL se configura con un factor de multiplicación de 6 mediante `PCTL=#$01E006`.
- La secuencia de comandos `movec #0,sp, move #$40,r6` y `move #-1,m6` resetean el **puntero de stack del hardware** y activan el **stack del software**.
- Mediante `IPRP=#$000003` se inicia el **sistema de interrupciones** para los puertos de audio y se establece el nivel de prioridad 2. En este curso no hacemos uso de las interrupciones del núcleo.
- **Puertos de audio**: los ajustes consisten en establecer el modo master con una frecuencia de 24,576 MHz, el modo de red con dos canales y los datos de 24 bits justificados a la izquierda, para un entorno de 32 bits, el I²S, las interrupciones para enviar y recibir, el tratamiento de las excepciones y el Last-Slot (canal derecho). Los registros

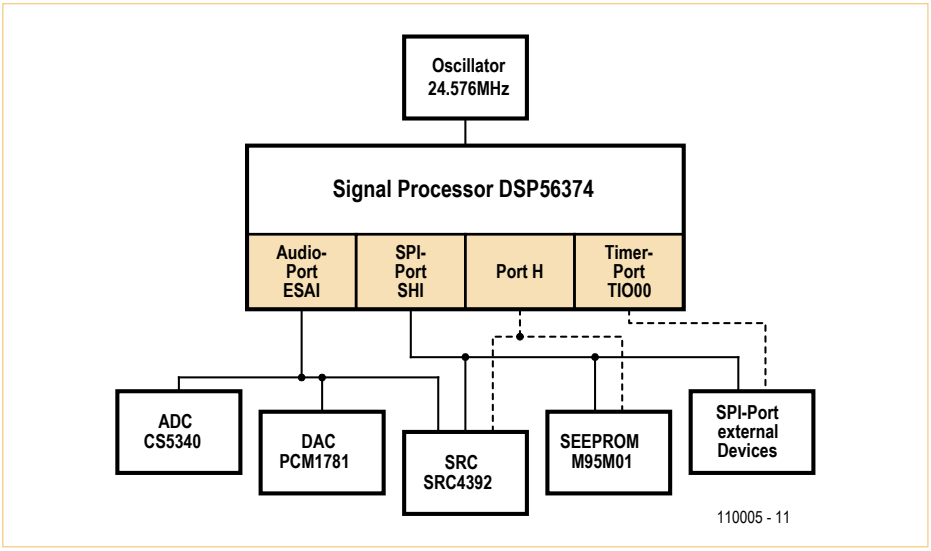


Figura 1. Sistema Solar del DSP.

de recepción y envío han de cargarse con los mismos datos `RCCR=TCCR=$FDD302` y `RCR=TCR=$D17D00`. Mediante `RSMA=RSMB=TSMA=TSMB=#$00FFFF` se configuran los canales en el modo de red (pueden ser hasta 32) individualmente para estar o no a disposición del emisor y el receptor. En la actual configuración se han liberado todos los canales posibles, de los cuales sólo utilizamos dos. Con `PCRC=PRRC=#$000FFF` se establece port C como puerto de audio. Es posible configurar también los pines individuales de port C como GPIO y utilizar con nuestro DSP como puertos de audio si fuera necesario.

- **Puerto SHI**: incluye las configuraciones del modo SPI, el funcionamiento como master a una frecuencia de 0,9216 MHz, sin interrupciones, `cpol=cpha=0`, narrow spike filter, FIFO off.

Programación del SRC: datos de 8 bits, ajustes `HCKR=#$002048` y `HCSR=#$000040`
Control del display de barras LED: datos de 16 bits, ajustes `HCKR=#$002048` y `HCSR=#$000044`

- Configuración del **puerto H**: éste se utiliza para la SRC y la SEEPROM así como para establecer el modo de boot.

PH4: entrada GPIO, Lock-Signal de SRC4392
PH3: salida GPIO, reset del SRC4392
PH2: funcionamiento como MODC-PIN, modo de boot del DSP
PH1: salida GPIO, ChipSelect del SRC SRC4392
PH0: salida GPIO, ChipSelect de la SEEPROM M95M01.
La configuración se lleva a cabo mediante `PCRH=#$000014` y `PRRH=#$00000F`.

3. Partes del programa antes de ejecutar el bucle de audio

- Permitir interrupciones, especialmente las de audio, de modo que el buffer pueda leerse o escribirse libremente y sea posible la sincronización controlada por flags de los datos de audio recibidos.
- Programación del SRC mediante el SHI en el modo SPI

4. ISR

- Las secciones del programa para el ISR se encuentran al final del archivo, que pertenecen al propio entorno de programa, y se encuentran en el archivo `esai4R2T.asm`.

"Audioloop"

El bucle de audio incluye el tratamiento digital de la señal de audio y está embebido en el entorno de programa. Seguimos el concepto de estructurar en bloques el tratamiento de la señal mediante subrutinas, según el cual cada una corresponde a un bloque independiente. La noción de tratamiento de señales en bloques ya ha aparecido en las técnicas analógicas y es bien conocida por los lectores.
Por consiguiente, nuestras subrutinas tienen *señales de entrada* y de *salida*. En el próximo ejemplo disponemos de cuatro señales, `SignalInL/R` y `SignalOutL/R`. Se diferencian, por ejemplo, de las de un generador de señales. En este caso las subrutinas sólo tienen señales de salida. Las señales de nuestro proyecto se representan fácilmente con una resolución de 24 bits y cada una ocupa una posición de memoria, en un área específica de la RAM del DSP. Luego están las subrutinas con parámetros que se ajustan según su modo de trabajo. En parámetros, se resumen los de configuración importantes para el tratamiento de la señal y los del programa del DSP, en conjunto. Dichos parámetros pueden incluir por ejemplo, las constantes de tiempo. La **figura 2** muestra una posible secuencia de subrutinas. Cuatro subrutinas, de `Subroutine A` a `Subroutine D`, se encargan de representar cuatro bloques de tratamiento de la señal. Los bloques cuadrados en el centro de la imagen representan las subrutinas que recorre el camino de la señal. Los bloques con bordes redondeados simbolizan los parámetros de dichas subrutinas. No todas las subrutinas necesitan parámetros, como ocurre en el caso de la codificada en C. El bloque con bordes rugosos contiene las señales, que circulan entre las subrutinas y simbolizan un "osciloscopio" para su observación en detalle. Estas señales tienen su propio espacio de memoria en el programa DSP y por lo tanto pueden revisarse en todo momento. Hemos implementado el tratamiento de la señal por bloques y desarrollado la posibilidad de detectar las características de las señales y si está funcionando dicho tratamiento. Podemos por ejemplo, tomar señales individuales de la cola del bucle de audio, en el buffer de salida y analizarlas con un osciloscopio de audio digital. Esto nos ayuda esencialmente a detectar errores en el tratamiento de la señal. El autor utiliza aquí una asequible tarjeta de sonido USB estándar, que puede conseguirse por bastante menos de 100 €, y un editor de Wav. Existen varios editores de Wav comerciales disponibles. Sin embargo, hemos conseguido editores de Wav y analizadores de audio gratuitos en Internet. Este software permite visualizar una señal (o varias) en el dominio del tiempo y el espectro en el dominio de la frecuencia. Utilizamos este software como si de un potente osciloscopio se tratase. En el espectro puede verse por ejemplo el factor de distorsión de un generador sinusoidal. Finalmente, podemos generar archivos de audio con el editor wav y procesarlos las veces que queramos. Los archivos de sonido pueden cargarse en software matemático como Matlab y tras algunos ajustes, analizar sus características. Un ejemplo útil es el test con un procesador dinámico con señales burst de duración, frecuencia y amplitud configurables. Ya que la tarjeta DSP ya incluye un DAC, podemos observar las señales con un osciloscopio analógico, lo cual se hace de un vistazo con un análisis de los datos más precisos posibles.
El ADC está conectado con SDI1(SDO4) y el SRC en SDI2(SDO3). La separación se hace en el buffer de entrada, que incluye su propia memoria.

```
x:RxBuffBase          ADC, linker Kanal
x:RxBuffBase+1 SRC.RX, linker Kanal
x:RxBuffBase+2 ADC, rechter Kanal
x:RxBuffBase+3 SRC.RX,  rechter Kanal.
```

El DAC y el coder AES3 con el emisor en el SRC se conectan mediante SDO0. En el buffer de salida, que dispone de dos posiciones de memoria, se hace la división:

```
x:TxBuffBase          DAC und SRC.TX, linker Kanal
x:TxBuffBase+1 DAC  und  SRC.TX, rechter Kanal.
```

Al principio y al final el bucle de audio incluye dos segmentos de programa para la sincronización del reloj de audio y para leer

el buffer de entrada, así como para escribir el de salida.

```
Audioloop
        jclr    #RightRx,x:LRFlag,*
        bclr    #RightRx,x:LRFlag
        move        x:RxBuffBase,a
; ADC CS5340, left
        move      x:RxBuffBase+2,b
; ADC CS5340, right
brset    #Lock_SRC4392,x:PDRH,NoSRC
; use ADC if SRC does not lock
          move     x:RxBuffBase+1,a
; SRC.RX SRC4392, left
          move     x:RxBuffBase+3,b
; SRC.RX SRC4392, right
NoSRC move     a,y:InL
        move     b,y:InR
```

Con las dos primeras líneas se lleva a cabo la sincronización, como se describe en la parte 2 del artículo. En la tercera y la cuarta línea se escriben las señales de ADC los registros acumuladores a y b. En la quinta línea se pregunta el estado del bit del Lock-Bit del SRC. Si el Lock-Bit indica una señal de audio válida, se escriben las señales SRC en dichos registros acumuladores a y b. En caso de que la señal no sea válida, se omite esta escritura. En la octava y novena línea se escriben las señales de los registros acumuladores en las posiciones de memoria `InL` y `InR`, para que sean leídas y procesadas por la primera subrutina de tratamiento de la señal. Mediante esta técnica nos aseguramos que siempre hay una señal de audio que leer. Si conectamos una señal digital a la tarjeta, ésta tendrá preferencia a la hora de ser leída. Si no existe ninguna señal digital válida, se procesa la del ADC.
El final del bucle de audio consiste en rellenar el buffer de salida con el contenido de los registros acumuladores a y b, que se cargaron previamente con las señales `OutL` y `OutR`, y el audioloop volverá a comenzar desde el principio.

```
        move     y:OutL,a
        move     y:OutR,b
        move      a,x:TxBuffBase    ;
-> DAC and SRC.TX, left
        move      b,x:TxBuffBase+1  ;
-> DAC and SRC.TX, right
        jmp      AudioLoop
```

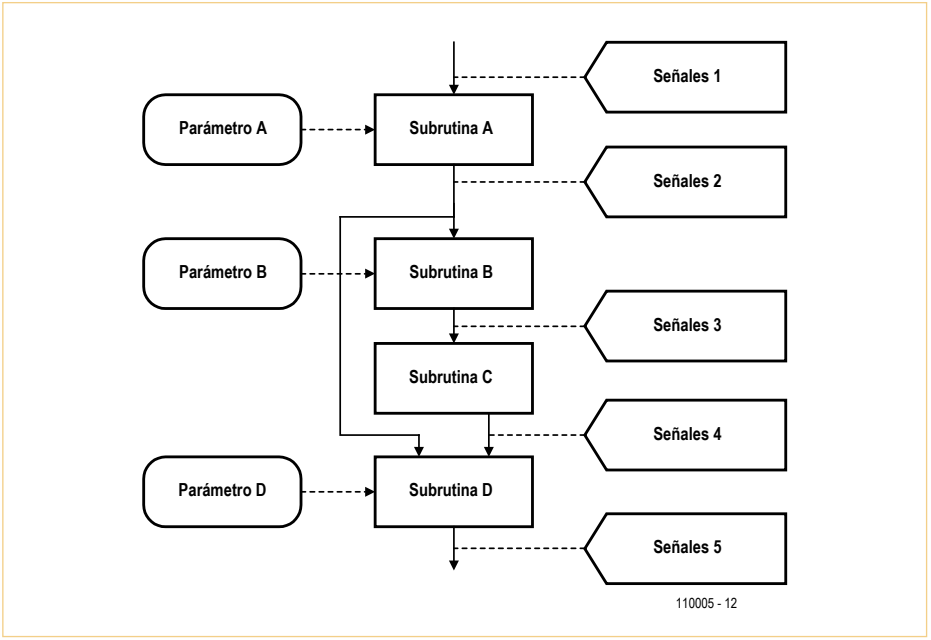


Figura 2. Subrutina de ejemplo en un bucle de audio.

Posiciones de memoria

La **tabla 1** muestra el uso de la memoria en el software actual. Hemos prestado menos atención a llenar los espacios de memoria, y más a mejorar la similitud con el programa actual. No en todos los programas se ocupan las áreas de memoria al completo.

Puntero de dirección, stack de software y contador de muestras

Dos de los ocho punteros de dirección, de `R0` a `R7`, se utilizan para “tareas de alto nivel” y por lo tanto no pueden ser utilizadas por el programa de usuario del DSP. El puntero `R6` es utilizado por el *s stack de software* en la X-RAM. Este stack de software complementa al stack de hardware y utiliza los espacios de memoria a partir de $40 en la X-RAM. Para el stack de software en nuestro programa están disponibles 32 posiciones de memoria, de $40 a $5F. En nuestro entorno de programa, el stack de software es utilizado por el ISR de audio, para cargar al puntero R0 con el atributo M0 y posteriormente ejecutar el reinicio del ISR. Hemos de tener en cuenta que al ISR no le está permitido manipular los registros que se utilizan en el programa. Este stack de software puede también utilizarse en programas del usuario para los registros del procesador, como los registros intermedios en el camino de los datos y sobre todo los de la AGU. El puntero `R7` se utiliza como módulo contador del tiempo total entre intervalos de muestreo. Utilizamos un módulo de 192 cuentas. Dicho contador, que lo llamamos sencillamente *Sample-Counter*, se utiliza dos veces. Primero, para leer las dos señales sinusoidales de 192 en el programa de test:

```
        Señal  senoidal,  f=1kHz,
x:$800..x:8BF
        Señal  senoidal,  f=2kHz,
y:$800..y:8BF
```

que posteriormente se entregan mediante el DAC o la salida digital. La segunda tarea consiste en controlar la escritura cíclica de los datos en el display LED mediante un medidor digital, en la segunda aplicación del DSP. El reloj de escritura corre a 48 kHz/192 = 250 Hz, lo cual significa que el display se rellena con nuevos datos cada 4 ms. En cualquier caso, el puntero `R7` podría utilizarse en cualquier programa de aplicación, si estando bloqueado por el stack éste lo libera nuevamente.

Subprogramas

Un subprograma, también llamado subrutina, tiene la siguiente estructura:

```
NombreDeLaSubrutina
        move  y:SignalInL,x0
        move  y:SignalInR,y0
        <Tratamiento de la señal,
si fuera necesario, dependiente de
unos parámetros>
        move  x0, y:SignalOutL
        move  y0, y:SignalOutR
        rts
```

El subprograma se invoca con el comando `jsr NombreDeLaSubrutina`. La instrucción `jsr` (jump to subroutine) indica al DSP, que tiene que revisar el contenido del registro de estado del procesador en el stack de hardware y ejecutar los comandos de la subrutina. La subrutina se cerrará mediante la instrucción `rts` (return from subroutine), es decir, el DSP revisará nuevamente el registro de estado del procesador y volverá al programa anterior. La entrega de parámetros a la subrutina puede hacerse mediante el stack de software.

Macros

Una macro DSP es sencillamente una macro de texto, que el ensamblador utiliza en el programa justo en el momento en que se le indica. Las macros tienen la siguiente estructura:

```
NombreDeLaMacro   macro   param1
param2 .. paramN

        endm
```

La llamada de una macro se lleva a cabo mediante el nombre de ésta y sus posibles parámetros. Con estas macros los programas pueden escribirse de forma más distendida, ya que éstas también son capaces de proporcionar parámetros. El ciclo de un programa no se altera con las llamadas constantes a una macro. Pero por el contrario, tenemos las subrutinas, que pueden ayudar a que el código sea más comprensible y reducido. Sin embargo, necesitamos ciclos de reloj del procesador y la utilización del stack para las llamadas y los retornos.

Inicializaciones para el tratamiento de señales

Directamente tras el inicio del bucle de audio, se ejecutan dos subprogramas:

- La subrutina `ZeroState`, mediante la cual las memorias de estado para el tratamiento de la señal se ponen a cero.
- La subrutina `SetDefaultParams`, con la cual se establecen los valores por defecto para los parámetros de proceso del tratamiento de la señal. En esta subrutina podemos cambiar los valores por defecto para adecuarlos a nuestras propias tareas. Esto también puede hacerse mediante el SPI con una unidad de entrada.

Carga e implementación de los programas

Una vez terminado un programa DSP, y tras comprobar que funciona, se ensambla. Esto puede hacerse mediante la ventana de CMD con la llamada

```
        asm56300 -a -b -l myprogram.
asm
```

Con la opción –l el ensamblador genera un archivo de listado que resulta muy útil para encontrar posibles errores. Ahora, el programa ensamblado `myprogram.cld` ya puede cargarse en el debugger conectado al DSP. Desde aquí podrá escribirse mediante un adaptador y ejecutarse en el propio DSP.

Consejos útiles

Para utilizar el ensamblador:

- El ensamblador interpreta el primer carácter del texto como etiqueta o marca. Una etiqueta comienza con una letra y no pueden utilizarse palabras clave como `move` o `a0`.
- El ensamblador DSP es de dos pasos. Un ensamblador de dos pasos coloca en la primera ejecución todas las etiquetas utilizadas y con referencia en una tabla de símbolos. En el segundo recorrido traduce las instrucciones y puede ensamblar debidamente ya que mediante la tabla de símbolos previa ya conoce el significado de las etiquetas, que de otro modo serían desconocidas en el momento de ensamblar.
- Son muy útiles en la programación de macros las etiquetas que comienzan con una barra baja _, pues indica que se trata de una etiqueta local.
- Al contrario que en las palabras clave, en las etiquetas se distingue entre mayúsculas y minúsculas.
- Los comandos DSP se escriben en segunda posición.
- El archivo `mioequ.asm` ha de integrarse mediante un `include` en el programa DSP, ya que dispone de numerosas abreviaturas del manual DSP para los registros de puertos.
- La dirección de programa más baja es p:$100, ya que en las anteriores se encuentran la tabla de vectores de interrupción y un espacio reservado.
- La instrucción org requiere la información de las áreas de memoria X, Y, P o L.
- Si el “pipeline” de instrucciones no se corta debido a desvíos en el programa o interrupciones largas, el tiempo de ejecución se reduce considerablemente, por lo tanto el ensamblador tiene que introducir de vez en cuando comandos de `nop` en el código objeto. En estos casos le indica al programador por medio de warnings que para que su código gane en rendimiento, debería implementar algo “con sentido” en lugar de las instrucciones de `nop`.
- En el manual del procesador encontramos al final algunas de las llamadas “programming sheets”, que simplifican la programación de los registros del procesador considerablemente. Recomendamos a cualquier programador que se haga con estas páginas, pueden imprimirse en cualquier momento desde el manual en PDF y ser utilizadas durante la fase de desarrollo, y posteriormente ser añadidas a la documentación.

Para programar el DSP:

- Las transferencias directas de valores (immediate moves) a posiciones de memoria como `move #$123456,x:$000100` no son posibles y se necesita utilizar un registro intermedio como por ejemplo `move #$123456,x0  move x0,x:$000100`. Para los registros de los periféricos en las direcciones altas de memoria pueden hacerse las transferencias del tipo `movep #$123456,x:$FFFFE0`.
- También con una transferencia de datos de una posición de memoria a otra han de utilizarse registros del procesador como “almacén temporal”, del tipo `move x:$000010,x0 move x0,y:$000010`. Sin embargo, con los registros de los periféricos sí son posibles las transferencias de este tipo.
- La instrucción `move #$F,x0` no nos conduce al resultado esperado `x0=$00000F`, sino a `x0=$0F0000`. Mediante `move #>$F,x0` obtenemos el resultado correcto. El DSP es un procesador fraccional, lo que significa que los números se almacenan desplazados a la izquierda.
- Los bucles de Do excluyentes han de referirse a etiquetas distintas. En adelante,

Tabla 1. Asignación de memoria para software del curso

Asignación de la X-RAM	Asignación de la Y-RAM	Rango	Asignación de la P-RAM
Buffer de audio RX	Señales	$00 a $0F	Tabla de vectores de interrupción y rango reservado
Buffer de audio TX	Señales	$10 a $1F	
Flags de audio y punteros	Punteros, coeficientes, memorias de estado	$20 a $2F	
Parámetros de programa		$30 a $3F	
Stack de software	Libre para utilizar	$40 a $4F	
Stack de software	Libre para utilizar	$50 a $5F	
Libre para utilizar		$60 a $9F	
Variables de ayuda		$A0 a $BF	
Secuencia de boot SRC	Libre para utilizar	$C0 a $CF	
Libre para utilizar		$D0 a $FF	
Coeficientes de filtros y polinomios		$100 a $5FF	Programas de usuario
Memoria circular, izquierda	Memoria circular, derecha	$600 a $7FF	
Señal senoidal a 1 kHz	Señal senoidal a 2 kHz	$800 a $8BF	
Libre para utilizar		$8C0 a $17FF	

las etiquetas han de separarse al menos mediante instrucciones de `nop`, sino existen ya otras instrucciones.

- La P-RAM no ha de llenarse con datos, ya que su acceso requería ciclos de procesador adicionales.
- Cuando han de multiplicarse números enteros, por ejemplo para cálculos de direcciones, es necesario dividir el resultado mediante el desplazamiento a la derecha `asr`, ya que el DSP con las multiplicaciones aplica automáticamente números fraccionales desplazados a la izquierda, y los enteros están por naturaleza desplazados a la derecha.
- Mediante instrucciones de dirección triples `mac` y `mpy` no pueden cubrirse el total de 16 combinaciones posibles de los cuatro registros de operando, ya que para la codificación de dichos registros sólo pueden utilizarse tres bits y no cuatro. Sin embargo, sí es posible hacer `mpy x0,x0,a` o `mac x0,y1,b`, pero no `mpy x1,x1,a` o `mpy y1,x0,b`.
- Las "fast interrupts" terminan con el comando `rti`.
- El puntero `R6` lo hemos reservado para el stack de software, que se encuentra en la X-RAM, en la dirección base X:$40. El stack de software lo necesitamos cuando toca manipular registros AGU en las subrutinas. Uno de los errores "famosos" consiste en forzar el direccionamiento por módulo en una subrutina de filtro, cuando esta orden no viene de fuera y en la propia subrutina aún no se ha concluido.
- Para comenzar el programa el puntero de stack `sp`, del stack de hardware, ha de inicializarse a cero.
- En aplicaciones particularmente críticas con el tiempo han de evitarse las ampliaciones de los comandos. En dichas extensiones, la segunda palabra opcional del comando consiste por ejemplo en una dirección o en un valor numérico. La instrucción `mpyi #0.3,x1,a` necesita una extensión para la representación fraccionada del número decimal 0.3 y consta por lo tanto de dos palabras. El comando `mpy x0,x1,a`, sin embargo, sólo necesita una palabra. Por ello, debemos guardar en el comienzo del programa por ejemplo las constantes en la RAM del DSP y si fuera necesario (posiblemente en una ejecución en paralelo) servirnos de un registro de operación, en este caso escribiéndolas en el registro de operando `x0`.
- Si consideramos un programa DSP, veremos que las instrucciones de `move` se utilizan muy a menudo, lo cual tiene su explicación, entre otras cosas porque el DSP es un procesador de registros. Sólo tenemos que servirnos de la habilidad del DSP para mover datos en paralelo. Generalmente esto significa que escribe registros (bastante) antes de utilizarlos, cuando puede hacerse de forma paralela a las operaciones aritméticas. Lamentablemente, los códigos de este tipo no son demasiado claros, ya que los segmentos de código correspondientes se distribuyen de forma espaciada.
- Tras algunos años programando DSPs hemos aprendido que los errores más comunes aparecen porque se utilizamos las mismas posiciones de memoria sin darnos cuenta. Por lo tanto, recomendamos listar detallada y completamente todas las posiciones de memoria utilizadas para cada programa DSP. Aparte, nuestra técnica incluye agrupar las variables de forma fija en un determinado rango, que quizá no es lo más eficiente de cara al uso completo de la memoria, pero sí menos susceptible de que aparezcan errores.
- Para los programas posteriores resulta útil escribir dos rutinas, mediante las cuales se escriben y se leen posteriormente 10 registros de datos `x0, y0, x1, y1, a0, b0, a1, b1, a2` y `b2` del stack de software. Podemos utilizar este stack para

Tabla 2. Diferencia entre números enteros con complemento a dos y fraccionarios

Entero	Decimal	Entero	Decimal	Fraccional	Decimal	Fraccional	Decimal
0000	0	1000	-8	0,000	0	1,000	-1,0
0001	1	1001	-7	0,001	0,125	1,001	-0,875
0010	2	1010	-6	0,010	0,25	1,010	-0,75
0011	3	1011	-5	0,011	0,375	1,011	-0,625
0100	4	1100	-4	0,100	0,5	1,100	-0,5
0101	5	1101	-3	0,101	0,625	1,101	-0,375
0110	6	1110	-2	0,110	0,75	1,110	-0,25
0111	7	1111	-1	0,111	0,875	1,111	-0,125

las subrutinas, y así evitar efectos secundarios debidos a la utilización de registros en el programa que se esté ejecutando.

Formatos numéricos de las señales en el DSP

Para las señales del DSP utilizamos sencillamente un formato numérico, con el cual los números binarios con coma se colocan inmediatamente después del primer bit, el *bit de signo*. Respecto de las cifras enteras, si este cambia su posición podremos distinguirlo dependiendo de si lleva coma o no. Este formato numérico se llama *complemento a dos fraccionario* o para resumir, sólo *fraccionario*. La diferencia entre los números enteros en complemento a dos y los fraccionarios puede verse de forma simple en cifras de 4 bits, de los cuales hay 16 números distintos, 7 positivos, 8 negativos y un cero, véase la **tabla 2**.
En la tabla se ha marcado el bit de signo por claridad y en los fraccionarios se incluye una coma. Los números se distinguen entre sí por sus posiciones, siendo en los enteros de 4 bits a la derecha del bit de signo, y de izquierda a derecha para los valores 4, 2 y 1, y en los fraccionarios tras la coma ½, ¼, $^1/_8$.
En el formato de enteros, las cifras se colocan hacia la derecha, y en el fraccional hacia la izquierda. Con ello, comprendemos fácilmente que en los enteros añadir ceros a la izquierda no cambia el valor para las cifras positivas, y añadir unos no lo cambia para cifras negativas. Con números fraccionarios esto tampoco ocurre si añadimos ceros a la derecha, tanto para cifras positivas como negativas. Un lector despierto se percatará de los valores límite para los números positivos y los negativos son distintos. Esto causa problemas a veces, puesto que nos solemos fijar en los límites de nuestro margen de actuación. Con un DSP de 24 bits la diferencia de valores de 2^{-23}= 1,1921·10^{-7}, en la mayoría de los casos no puede obviarse.
Ahora explicamos el motivo de utilizar números fraccionarios. Supongamos que disponemos de un DSP cuya precisión simple es de números de 4 bits y la doble es de 8 bits, que sin embargo, con un DSP de 24 bits será interpretada como una resolución muy reducida (esto es la distancia entre dos cifras consecutivas). Ahora, digamos que este DSP tiene que implementar una *recuantización*, lo cual en el tratamiento de señales digitales es una operación bastante común. En el ejemplo un número de 8 bits tiene que ser recuantizado a número de 4 bits. El de 8 bits es:

```
0,100 0100  Valor numérico: ½ + 1/32
= 0,53125
```

En este ejemplo la recuantización se limita a *prescindir* de los cuatro valores inferiores, con lo que la cifra queda

```
0,100       Valor numérico: ½ = 0,5
```

Esto no es muy impresionante que digamos, sin embargo, el lector debería considerar qué pasaría si quisiéramos hacer esta tarea con números enteros.
De vuelta al DSP: Este DSP realiza cálculos con resolución simple de 24 bits y doble de 48 bits. Sabemos que en el audio digital se utilizan básicamente 24 bits, lo cual corresponde en el DSP a la resolución simple. Pero, ¿qué ocurre si multiplicamos dos valores de la señal entre sí, o queremos aplicar una atenuación de 20 dB? En ambos casos la multiplicación da como resultado un valor con una resolución doble, de 48 bits. Consideremos esto en el DSP simplificado de 4 bits. Para multiplicar, tendríamos dos números:

```
0,010 * 0,001  Valores numéricos:
0,25 * 0,125 = 0,03125
0,000 0100     Valor numérico:
0,03125
```

El resultado tiene ocho posiciones.
Uno de los consejos de arriba se ilustra bien con el ejemplo. Cuando multiplicamos entre sí números enteros, por ejemplo, para calcular las direcciones de las posiciones de memoria, el resultado del multiplicador del DSP ha de ajustarse mediante una operación de desplazamiento. Nos centramos en este último ejemplo, interpretando los números de otra forma:

```
0010 * 0001    Valores numéricos: 2
* 1 = 2
0000 0100      Valor numérico:
4, resultado de un fraccionario
multiplicado.
0000 0010      Valor numérico: 2,
resultado correcto tras el despla-
zamiento aritmético de una posición
a la derecha.
```

Próximamente...

Hasta aquí sobre el la estructura de los programas DSP. En la sexta entrega utilizaremos la tarjeta DSP como generador digital de señales de audio.

(110005)

Monitor de tensión de bajo consumo

Rolf Blijleven (Holanda)

La electrónica autónoma alimentada por células fotovoltaicas es bastante fácil de hacer, siempre y cuando no tenga que funcionar continuamente. También puedes almacenar el escaso rendimiento de una célula hasta que tengas suficiente para alimentar el circuito durante un momento. ¿Cómo sabes si tienes suficiente? ¡Con un monitor de tensión! El problema: que también consume energía. Pero no debe consumir nada de la energía almacenada – o en todo caso lo menos posible. ¡Unos pocos microamperios marcan ya la diferencia!

En enero de este año tratamos el principio de cosechar energía. El desafío es hacer funcionar un circuito con una pequeña fuente de energía, sin una alimentación externa, sin pilas. En este caso la fuente de energía es una pequeña célula fotovoltaica (5 euros), que genera una tensión máxima de 1 voltio con 100 mA. Una célula así sólo genera ese valor máximo en un día con un cielo despejado y si la célula se encuentra colocada perpendicularmente en comparación con la posición del sol. Con el cielo cubierto se queda sólo en un par de milivoltios con unas decenas de microamperios. Sin embargo, es posible hacer algo útil con tan poca energía - no continuamente, pero sí brevemente durante un par de veces al día.

Configuración

La **Figura 1** muestra el esquema de bloques. La salida de la célula fotovoltaica está conectada a un multiplicador de tensión que carga el condensador de almacenamiento Cs. En el Elektor de enero de este año tratamos varios multiplicadores de tensión, por eso lo reflejamos aquí como bloque funcional. El amplificador de este esquema de bloques es un comparador alimentado por el condensador de almacenamiento. En la práctica encontrarás varias resistencias alrededor. Si la entrada + alcanza la tensión umbral, la salida del comparador cambia a nivel alto hacia la tensión de almacenamiento Vs y el transistor se abre. Después de que la salida del operacional haya vuelto a su estado inicial, Rt y Ct hacen que el transistor quede abierto un poco más de tiempo. Debido a esto, la carga queda conectada mientras dure la energía almacenada en Cs. A esta construcción también se la llama una *solar engine*. Puedes encontrar más información sobre esto en Internet ([1] entre otros sitios).

En una mañana cubierta la corriente de carga sólo asciende desde unos 20 hasta 30 µA, así que el condensador de almacenamiento se carga muy lentamente. Sin embargo, tenemos que verificar continuamente si ya hay suficiente energía almacenada – no podemos utilizar un temporizador, porque no disponemos de pilas. Los

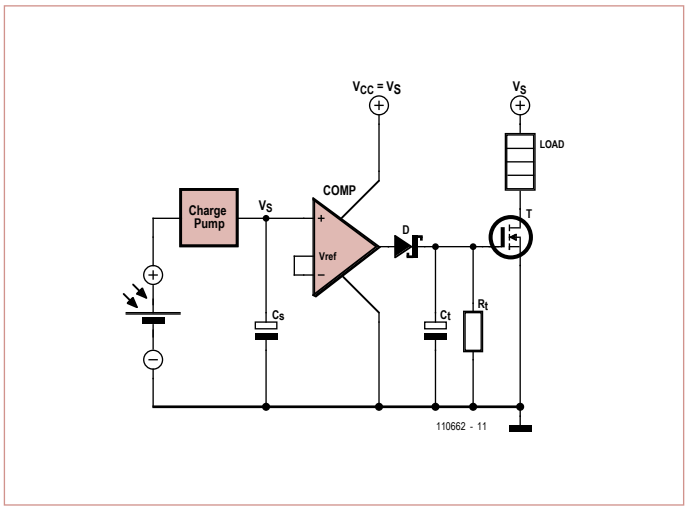


Figura 1. Esquema de bloques común de un circuito de cosechar energía.

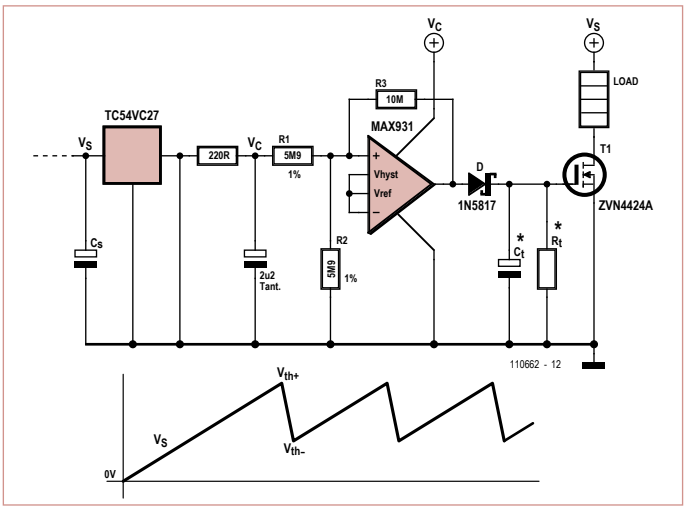


Figura 2. Esquema del monitor de tensión de bajo consumo. Todos los semiconductores están disponibles como componentes SMD y de taladros pasantes.

experimentos con el monitor de tensión MAX8212 de Maxim [2, 3] como comparador funcionaron, pero no hubo plena satisfacción. Este integrado con las resistencias de ajuste consume unos 6 µA – más de una cuarta parte de la corriente de carga en una mañana cubierta.

Solución práctica

En la **figura 2** se refleja una mejor solución. El TC54VC27 es un detector de tensión. Este integrado con 3 terminales consume menos de 1 µA mientras Vs no alcance aún la tensión umbral de 2,7 V, y eso es durante la mayor parte del ciclo de carga.
Por encima de este umbral el TC54 pasa simplemente a la salida lo que hay en su entrada. Este cambio se debe hacer de forma controlada ya que la salida de TC54 también provee de alimentación al comparador, un MAX931 de Maxim (consumo en reposo 4 µA). Si este recibiera bruscamente su alimentación, podría ocurrir que su salida se pusiera directamente a nivel alto, de modo que el transistor condujera brevemente y Vs bajase un poquito. De esta forma Cs nunca alcanzaría la tensión deseada; eso no es lo que queremos, así que evitamos este comportamiento con el circuito RC de 220 Ω y 2,2 µF. El MAX931 se queda desconectado hasta alcanzar la tensión umbral Vth+, determinada por R1, R2 y R3 (3,1 voltios con los valores dados). La diferencia con 2,7 voltios no parece muy elevada, pero si es substancial. Notarás esto cuando la carga sea un motor. En la hoja de características del MAX931 [4] se explica el cálculo de R1 y R2. El monitor tiene un consumo máximo de 5 µA entre 2,7 voltios y la tensión umbral.

Hay integrados del tipo TC54 con diferentes tensiones umbrales incorporadas de entre 1,4 y 7,7 voltios, ver la hoja de características [5]. Y efectivamente, podíamos haber optado por uno con una tensión umbral más próxima por debajo de Vth+. A lo mejor te preguntas si el comparador es realmente necesario. La respuesta es sí, ya que el transistor tiene que conectarse de forma claramente definida. En este caso el TC54 empieza a conducir aproximadamente a los 2,75 voltios. Si conectamos un MOSFET directamente detrás del TC54, entonces empezaría a conducir un poquito, la tensión sobre Cs bajaría, el TC54 volvería a desconectarse y así nunca empezaría a funcionar la cosa. Una resistencia de realimentación positiva entre la salida y la entrada del TC54 no es ninguna solución, ya que ésta cargaría a Cs.
La constante de tiempo de Ct con Rt determina cuanto tiempo queda abierto el transistor. No hace falta que Cs se vacíe más de lo necesario para que la carga pueda hacer su trabajo. Supón que la carga junto con R_{dsON} del MOSFET asciende a 25 Ω y que Cs consta de 3 condensadores electrolíticos de 4700 µF, o sea 14,1 mF en total. Entonces Cs se descargaría en 0,35 segundos. Pero si 0,2 segundos es suficiente, Cs puede quedarse cargado con una tercera parte, así que la próxima vez se llenará antes. Si tomas para Ct = 2,2 µF, entonces puede ascender a Rt = 100 kΩ. En la gráfica está reflejado el desarrollo de tensión en tiempo de Vs.

Parece que las diferencias sean mínimas, sin embargo son claramente visibles. No tenía ganas de mirar continuamente a un contador durante las pruebas del circuito.

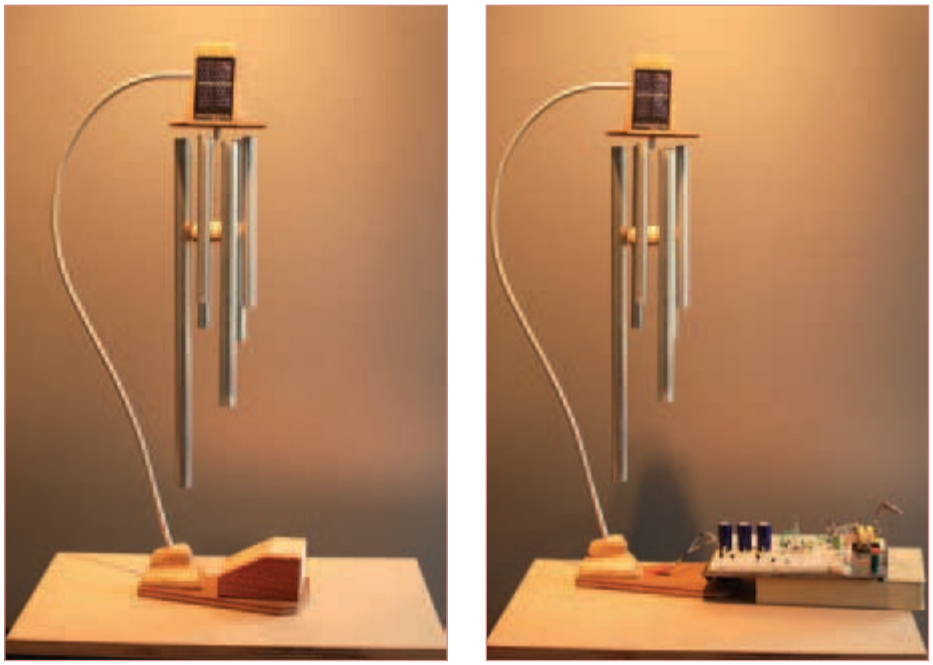


Figura 3. Suena un juego de campanas cuando se ha cosechado suficiente energía. Así no hace falta mirar cada vez al contador.

Así que construí un juego de campanas con un motor eléctrico, para poder oír cuántas veces se llena el almacén de energía (y qué tiempo hace fuera). Con este monitor de bajo consumo también suena varias veces al día si el tiempo es lluvioso. Con diseños menos ahorradores permanece mudo muchas veces con un tiempo así.

(110662)

Enlaces web

[1] http://library.solarbotics.net/circuits/se.html

[2] http://www.maxim-ic.com/datasheet/index.mvp/id/1273

[3] http://www.iamwhen.com/archives/53-Herbert-1701-Species-B-Generation-1.html

[4] http://www.maxim-ic.com/datasheet/index.mvp/id/1219

[5] http://ww1.microchip.com/downloads/en/DeviceDoc/21434h.pdf

Sencillo detector de murciélagos

Barato, sensible y fácil de construir

Jan van Eck (Holanda) j.vaneck@fontys.nl

Muchas asociaciones de animales de toda Europa han proclamado el año 2011 como el año del murciélago. De esta forma quieren llamar la atención de este mamífero volador, para mucha gente, desconocido y bastante misterioso. Ciertamente los murciélagos se encuentran en este momento en hibernación, pero esto te permite construir ahora este sencillo detector e irte de caza de sonidos de murciélagos la próxima primavera.

Para percibir los murciélagos a través de la audición hemos recurrido a algunos trucos electrónicos. Los murciélagos utilizan ultrasonidos que se encuentran muy por encima del alcance de nuestro oído humano para la detección de objetos. Las frecuencias producidas por muchas especies de murciélagos se encuentran alrededor de los 40 kHz y ese es también el alcance exacto de la mayoría de los transductores de ultrasonido estándares.

Si amplificamos las señales captadas por uno de estos transductores y las pasamos por un divisor de frecuencias, llegarán al alcance audible y podremos oírlas.

El circuito

Hemos optado por un receptor de ultrasonidos de 40 kHz del tipo 400SR160, porque la mayoría de los murciélagos que hay a nuestro alrededor producen sonidos de cerca de 40 kHz. El autor optó por una versión barata de plástico de este transductor, que se puede dotar de protección fácilmente con un folio de aluminio o con cinta adhesiva metalizada.

A continuación, IC1, un LM386 (**figura 1**), amplifica la señal captada por el transductor unas 200 veces. En realidad, se trata de un integrado con una etapa de potencia, pero por su bajo precio puede servir perfectamente como etapa de amplificación 'normal'. Tiene la gran ventaja de que no requiere componentes adicionales, excepto por algunos componentes de desacoplo. Colocando un condensador sobre la realimentación interna del LM386 (terminales 1 y 8) puedes aumentar la amplificación mínima de 26 dB a 46 dB. En la aplicación estándar esto es un condensador electrolítico de 10 µF, pero sólo se necesita este valor si se quiere amplificar uniformemente el ancho de banda de audio completo. No es necesario hacer esto en esta aplicación de modo que se puede reducir el condensador hasta 220 nF (C1). Así la frecuencia de corte se queda en unos 4 kHz. En principio, el condensador podría ser

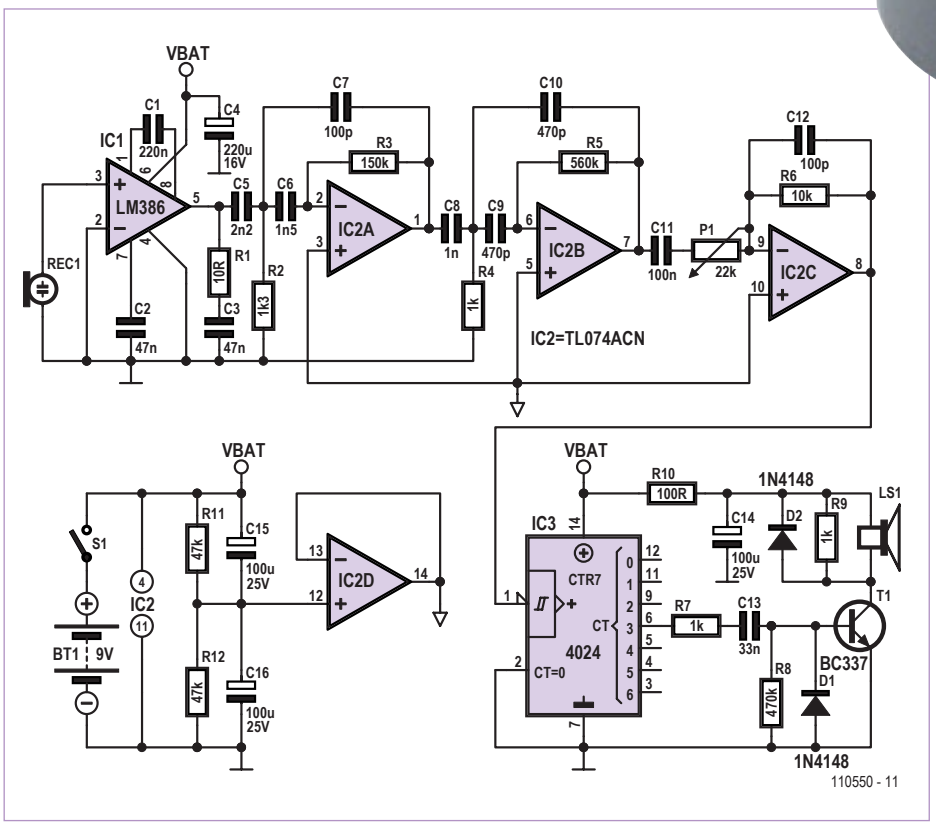
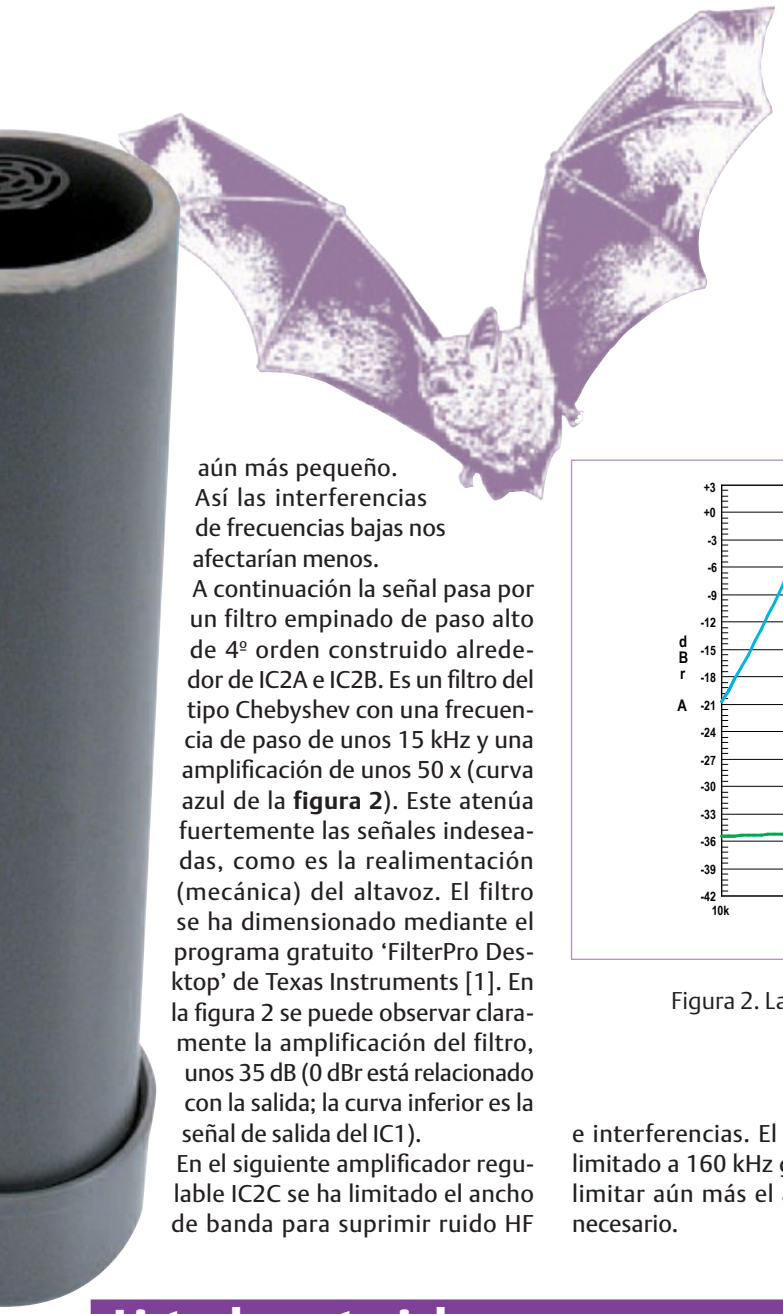


Figura 1. Esquema del sencillo detector de murciélagos.



aún más pequeño. Así las interferencias de frecuencias bajas nos afectarían menos.

A continuación la señal pasa por un filtro empinado de paso alto de 4º orden construido alrededor de IC2A e IC2B. Es un filtro del tipo Chebyshev con una frecuencia de paso de unos 15 kHz y una amplificación de unos 50 x (curva azul de la **figura 2**). Este atenúa fuertemente las señales indeseadas, como es la realimentación (mecánica) del altavoz. El filtro se ha dimensionado mediante el programa gratuito 'FilterPro Desktop' de Texas Instruments [1]. En la figura 2 se puede observar claramente la amplificación del filtro, unos 35 dB (0 dBr está relacionado con la salida; la curva inferior es la señal de salida del IC1).

En el siguiente amplificador regulable IC2C se ha limitado el ancho de banda para suprimir ruido HF e interferencias. El ancho de banda está limitado a 160 kHz gracias al C12. Puedes limitar aún más el ancho de banda si es necesario.

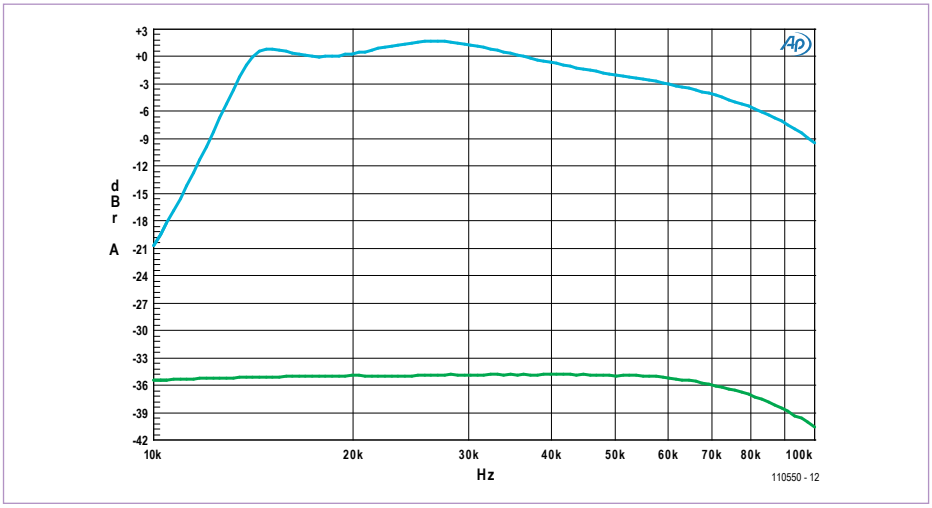


Figura 2. La respuesta en frecuencia en la salida de IC2B (parte superior) y en la salida de IC1 (parte inferior).

Con el divisor de tensión de 2 x 47 k (R11/R12) puesto a la mitad de la tensión de alimentación y con los operacionales restantes del TL074 se ha creado

Lista de materiales

Resistencias (0,25 W, 5 %):
R1 = 10 Ω
R2 = 1k3
R3 = 150 kΩ
R4,R7,R9 = 1 kΩ
R5 = 560 kΩ
R6 = 10 kΩ
R8 = 470 kΩ
R10 = 100 Ω
R11,R12 = 47 kΩ
P1 = 22 kΩ potenciómetro de ajuste, instalación vertical

Condensadores:
C1 = 220 nF MKT, paso 5 mm
C2,C3 = 47 nF MKT, paso 5 mm
C4 = 220 µF/16 V radial, paso 2,5 mm
C5 = 2n2 MKT, paso 5 mm
C6 = 1n5 MKT, paso 5 mm
C7,C12 = 100 pF cerámico, paso 5 mm
C8 = 1 nF MKT, paso 5 mm
C9,C10 = 470 pF cerámico, paso 5 mm
C11 = 100 nF MKT, paso 5 mm
C13 = 33 nF MKT, paso 5 mm
C14,C15,C16 = 100 µF/25 V radial, paso 2,5 mm

Semiconductores:
D1,D2 = 1N4148
T1 = BC337-40
IC1 = LM386N-3
IC2 = TL074CN
IC3 = 4024

Varios:
REC1 = receptor de ultrasonido 40 kHz (por ejemplo Prowave 400SR16P, diámetro 16 mm)
LS1,S1,BT1 = conector de 2-terminales, paso 2,54 mm
Conector 3 x 2-terminales para conectar el altavoz, interruptor y pila
LS1 = altavoz 8 Ω/0,3 W, diámetro 20 mm (por ejemplo Kingstate KDMG20008)
S1 = interruptor de deslizamiento 1 x SPST
BT1 = pila de 9 V + conector
Placa 110550-1 (ver www.elektor.es)

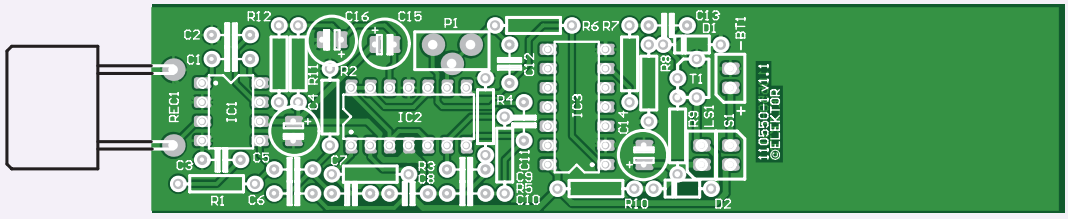
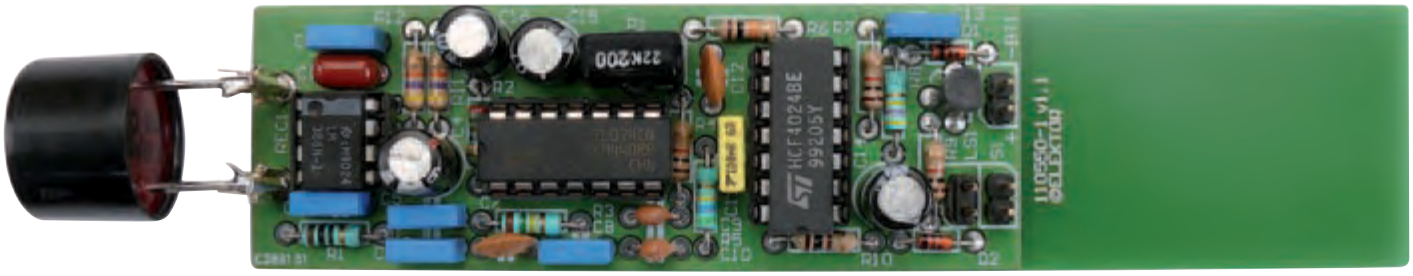


Figura 3. La placa del circuito es larga y estrecha para que quepa en un trozo de tubo de PVC.



un punto de masa virtual para los demás operacionales.
A continuación se entrega la señal de IC2C a la entrada de reloj de un 4024, un contador binario de 7 etapas. En la salida CT3 (terminal 6) está disponible la señal divida por 16. Así surgen pulsos en el área audible (2...3 kHz), que un altavoz convierte en audio a través de T1. Se ha añadido D1 como protección contra tensiones negativas en la base del transistor. R7 limita la corriente que tiene que proveer IC3 a T1. Debido al carácter inductivo del altavoz, se ha colocado en paralelo a LS1, un diodo de protección que protege T1 contra altos picos de inducción. Para LS1 se puede utilizar también un altavoz piezoeléctrico. Se ha añadido R9 para garantizar su buen funcionamiento, ya que por lo contrario no se oiría nada (de hecho los zumbadores piezoeléctrico son altamente capacitivos). Para suprimir interferencias en la tensión de alimentación, se ha desacoplado adecuadamente la tensión al altavoz mediante R10 y R14. De hecho la entrada es especialmente sensible. El consumo del circuito completo asciende a unos 14 mA en reposo, durante la recepción de ultrasonidos asciende a un máximo de 90 mA. El circuito funciona bien aún a unos 4,5 V.

Miniplaca

En la **figura 3** puedes ver el diseño desarrollado para la placa para este detector de murciélagos. En la parte inferior se puede montar una pila de 9 V. Esta placa está provista de masa en la cara superior que sirve de protección en relación a la sensibilidad bastante elevada.
La placa tiene tales dimensiones para que quepa en un tubo de PVC estándar con un diámetro interno de 33...35 mm, a la venta en cualquier ferretería. Ahí también se puede comprar una tapa que encaje bien, en la que se pueda montar un pequeño altavoz y un interruptor (ver **figura 4**). Si se provee a la parte inferior de 3 patas de goma adhesivas, podrás colocar el tubo verticalmente en una mesa de jardín y escuchar inmediatamente a los murciélagos cuando lo sobrevuelan.

Con la ayuda de un potenciómetro de ajuste se puede ajustar la sensibilidad (ajustarlo para que no salga ningún sonido o que salga un sonido golpeando mínimamente del altavoz). Asegúrate de que no haya ninguna fuente de ultrasonido, como son: los armarios de tubos fluorescentes, televisores, pantallas, fuentes de alimentación conmutadas, etc. Estas se detectan a una distancia de varios metros.
El simple movimiento de frotar con el dedo pulgar e índice, el crujir de una bolsa de plástico o hacer sonar un llavero, genera ya suficiente ultrasonido como para que el circuito lo detecte a una distancia de varios metros. Los murciélagos son bastante ruidosos y pueden ser detectados a una distancia de más de 30 m con este circuito. También se pueden percibir los diferentes sonidos que genera un murciélago.

(110550)

Enlaces Web

[1] www.ti.com/tool/filterpro

Figura 4. En la tapa se han montado un interruptor de conexión/desconexión y un altavoz.

Trabajar con plantillas de estarcido

Thijs Beckers (Redacción Holanda/laboratorio de Elektor) y Antoine Authier (laboratorio de Elektor)

La creación de placas impresas no supone ningún problema para la mayoría de los electrónicos. Sin embargo, queremos enseñaros un método que muchos electrónicos probablemente no lo han visto aún, es decir la aplicación de pasta de soldar con ayuda de una plantilla de estarcido, más conocido como “esténcil”.

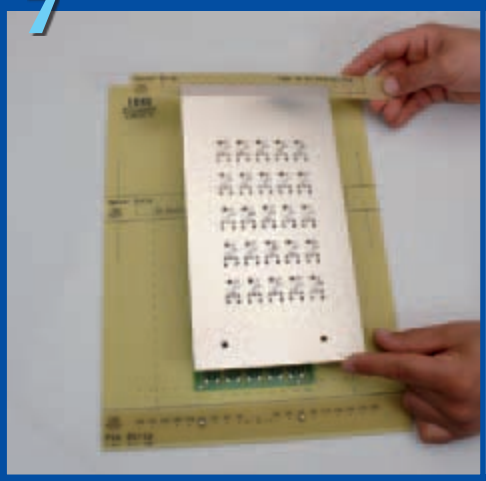
Muchos fabricantes de placas impresas ofrecen la opción de encargar un esténcil con el pedido de una placa (panel) que contenga principalmente componentes SMD. La intención de este esténcil es que la aplicación de la pasta de soldar sea más sencilla, más rápida y mejor. Un muy buen ejemplo de esto es la placa panel de Elektor BOB.

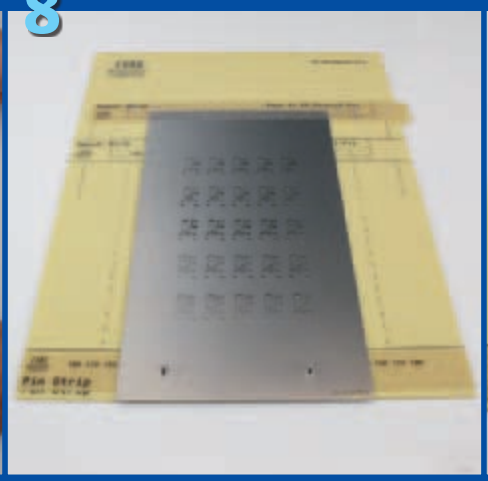
Figura 1
Estos son los compontes que recibes con el kit de esténcil: la placa (delante), el propio esténcil (a la derecha) y un soporte con material de fijación (a la izquierda).

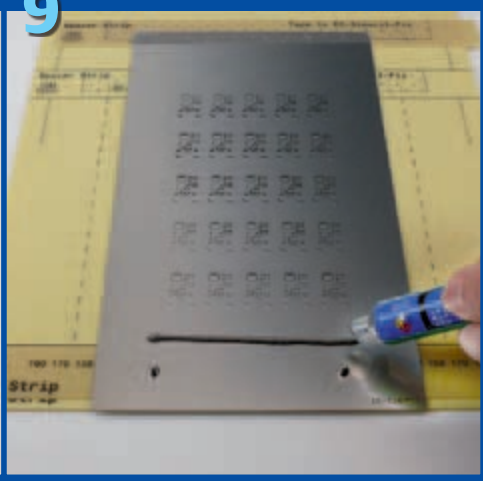
Figura 2
El esténcil y la placa se mantienen en su sitio gracias a estos pines con formato especial.

Figura 3
Para eso pinchamos primero las partes con forma de cono sobre la tira del material de la placa, y a continuación se coloca en el portaplacas.

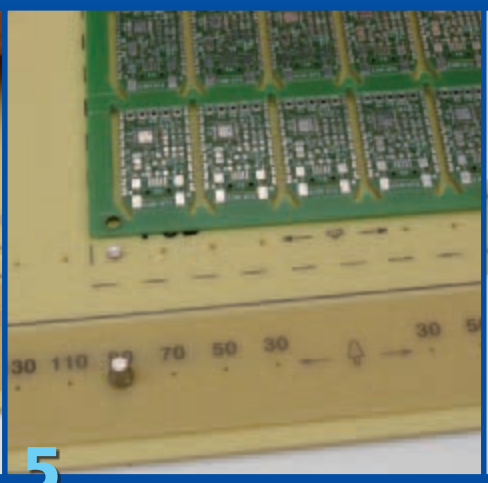
Figura 4
Después colocamos las clavijas del portaplacas en el portaplacas.







4



5

6

Figura 5
Ahora puedes fijar la placa impresa en el portaplacas. Asegúrate de que la placa esté bien limpia. Las manchas de dedos u otras contaminaciones en los islotes son inadmisibles.

Figura 6
Tenemos que asegurarnos de que se puede colocar bien el esténcil sobre la placa impresa (sin ángulos doblados). Para eso pegamos la parte superior del esténcil en un trozo de placa que tenga el mismo grosor que la placa impresa.

Figura 7
Ahora colocamos el esténcil sobre la placa impresa.

Figura 8
Las clavijas en forma de cono mantienen el esténcil perfectamente en su sitio. Puedes ver que los islotes se quedan sin cubrir.

Figura 9
Ahora puedes aplicar la pasta de soldar. Sólo vamos a aplicarla sobre la primera fila de la placa, así que no suministramos demasiada pasta.

Figura 10
A continuación tenemos que repartir en un ¡único! movimiento la pasta de soldar con una regla de goma. Tira de la regla de goma hacia ti, esto es lo que mejor resultado da.

Figura 11
Si todo se ha hecho bien, la pasta de soldar estará perfectamente aplicada al retirar el esténcil.

Figura 12
En primer plano vemos que incluso a los islotes más pequeños del integrado ha llegado la dosis exacta en el lugar exacto – ¡la distancia entre los terminales del integrado asciende a tan sólo 0,5 mm!

La placa está ahora lista para recibir los componentes. Y cuando estos estén colocados puedes meter todo en el horno de refusión, pero esto es otra historia.

(110514)

10

11

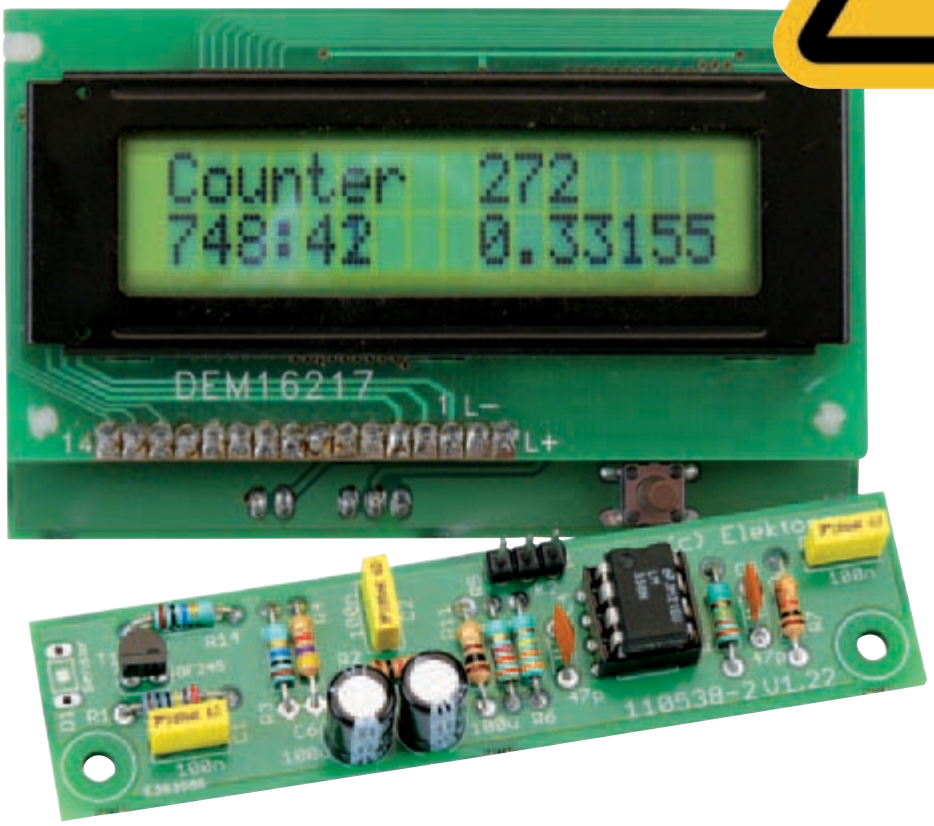
12

Medidor de radiación mejorado

Contador de alfa, beta y gamma

Burkhard Kainka (Alemania)

Para medir la radiación no necesitamos más que un fotodiodo PIN y el amplificador de instrumentación adecuado. Aquí presentamos un preamplificador optimizado con un contador por microcontrolador. Dicho controlador también se encarga de la coordinación de las medidas y muestra la tasa de pulsos en “counts per minute”.



El dispositivo puede utilizarse con distintos sensores para medir la radiación gamma y alfa. Es apto para medidas durante largo tiempo y el análisis de probetas de baja actividad. En comparación con un contador Geiger, un fotodiodo ofrece valores de cero más bajos, debido a su reducido tamaño, y la radiación de pequeñas muestras se aprecia con mayor facilidad.
Otra ventaja del sensor semiconductor es que permite medir la energía de cada partícula por separado. Así, podemos testear probetas de forma más precisa que con un contador Geiger. Mediante un software de PC opcional puede obtenerse el espectro energético y sacar conclusiones del objeto a examen.

Preamplificador

En la edición de Elektor de junio de 2011 los intentos con el fotodiodo BPW34 como detector gamma [1] no resultaban tan simples, ya que proporcionaba pulsos extremadamente cortos. Ahora, este amplificador optimizado se encargará de que los pulsos sean audibles incluso sin hacer uso de un comparador. Los amplificadores de señal a la entrada que utilizan un JFET BF245B y los amplificadores operacionales posteriores ofrecen en total una ganancia en tensión de aproximadamente 30.000. A la salida tenemos pulsos de hasta 200 mV con una duración de 0,5 ms, que pueden oírse sin necesidad de circuitería adicional y procesarse mediante un contador.
El circuito (ver la **figura 1**) puede equiparse con multitud de fotodiodos en paralelo. Así se incrementa la tasa de pulsos. Al mismo tiempo se reduce la tensión de la señal,

Productos y servicios Elektor

- Tarjeta 110538-1
- Kit (componentes y tarjeta) 110538-71
- Tarjeta puente USB-FT232R 110553-91
- Descarga gratuita del diseño en PDF en [2]
- Descarga gratuita del software y firmware (archivo 110538-11 en [2])

Características

- Mide la radiación α, β y γ
- Fácil montaje con componentes estándar
- Conexión con el PC mediante la tarjeta puente USB-FT232R de Elektor
- Umbral de respuesta (Threshold) configurable por software (vía PC)
- Puede utilizarse con varios tipos de sensores

ya que entra en juego una mayor capacitancia. Esto también recorta la sensibilidad ante señales débiles, lo cual tiene la ventaja de filtrarlas parcialmente.

El JFET ofrece una buena relación ruido-señal con alta resistencia de entrada. En la resistencia de la fuente del BF245B tendremos una tensión continua de entre 2 y 3 V, especialmente independiente de la tensión de alimentación (más en la versión BF245C que en la A). Así obtendremos un punto de trabajo apto para el operacional. El fotodiodo funciona con la máxima tensión de alimentación, ya que por encima de los 20 MΩ la tensión de la puerta se hace cero. Esto resulta especialmente importante, ya que la capacidad del diodo cae con tensiones crecientes.

El contador

El contador de pulsos se ha realizado mediante un ATmega88 y un LCD de dos

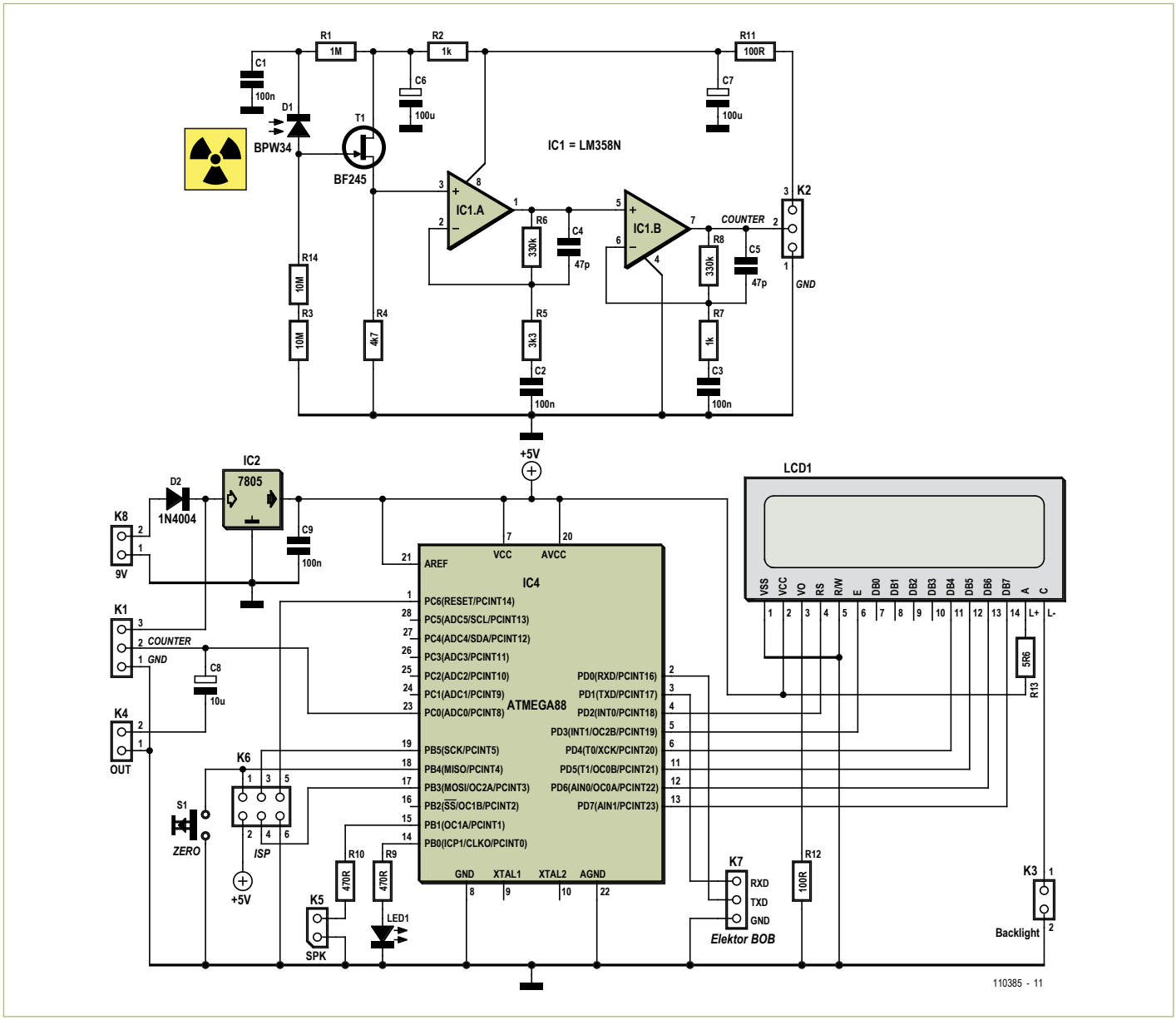


Figura 1. Esquema de circuito del preamplificador y la tarjeta controladora.

Lista de materiales

Resistencias:
R1 = 1 MΩ
R2, R7 = 1 kΩ
R3, R14 = 10 MΩ
R4 = 4k7
R5 = 3k3
R6, R8 = 330 kΩ
R9, R10 = 470 Ω
R11, R12 = 100 Ω
R13 = 5Ω6

Condensadores:
C1, C2, C3, C9 = 100 nF
C4, C5 = 47 pF
C6, C7 = 100 µF/ 16V
C8 = 10 µF/ 16V

Semiconductores:
D1 = BPW34
D2 = 1N4001
D3 = LED de 5 mm, verde
IC1 = ATmega88PA-PU (Atmel), programado
IC2 = LM358N
IC3 = 78L05
T1 = BF245B

Varios:
S1 = pulsador de 1 contacto
K1 a K8 = conector de pines, por ejemplo TE-Connectivity 3-826926-6
LCD1 = DEM16217, disponible en Elektor (030451-72)

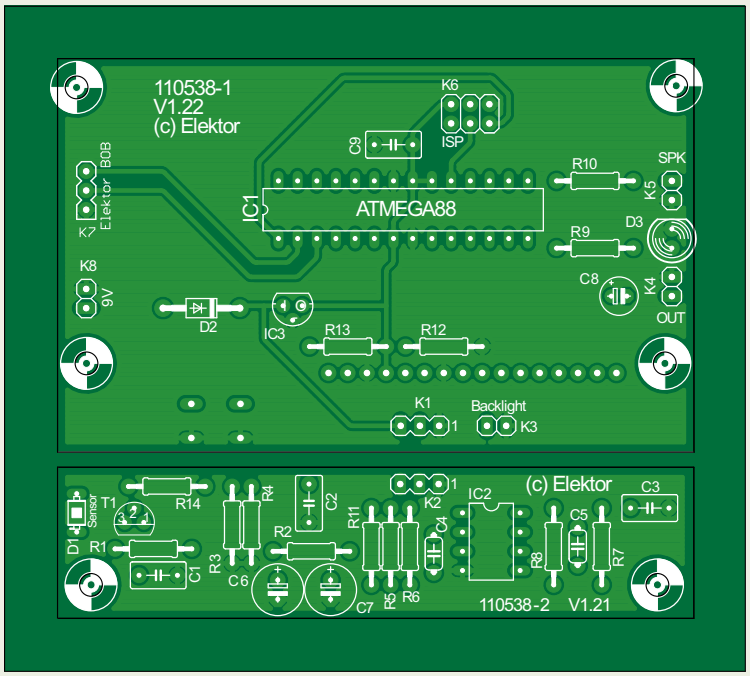


Figura 2. La tarjeta del medidor de radiación.

líneas. La tensión de alimentación de 9 a 12 V se suministra a través de D1, que actúa como protección contra tensiones inversas para el regulador IC2 (78L05), que provee con 5 V al microcontrolador y al sensor. La tarjeta dispone de un conector ISP para mayor comodidad en las actualizaciones de software y otro para el puerto serie (K1), mediante el cual pueden leerse o transmitirse datos al PC. Las señales RXD y TXD tienen nivel TTL y son aptas para la tarjeta puente USB-FT232R de la edición se septiembre de 2011 de Elektor.

K1 conecta con el amplificador de instrumentación. La reducida tensión continua de la salida y las señales útiles van directamente a la entrada analógica ADC0 del controlador. En el inicio el software del controlador tiene que determinar el nivel medio de offset. Una vez definido, se contarán los pulsos que superen dicho nivel. El interruptor S1 sirve para comenzar una nueva medición sin tener que determinar otra vez un nivel de offset nuevo. Así, es posible calcular el nivel de cero primero sin la muestra a medir, y luego con la probeta comenzar a tomar los valores de medida reales.

Cada pulso contado genera una señal de salida para LED y K5, en donde puede colocarse un pequeño altavoz. De este modo, la cuenta puede comprobarse de forma tanto óptica como acústica. El altavoz de 8 a 32 Ω también puede conectarse a su vez a un control de volumen (potenciómetro logarítmico de 1 kΩ), pues en mediciones largas el crujido del contador puede volverse molesto.

La señal del sensor sin procesar también está disponible, adicionalmente, mediante C8 en la salida K4, y desacoplada, por ejemplo, gracias a un conector BNC, en el cual puede conectarse un osciloscopio. Si aquí conectamos un amplificador de audio, podrá escucharse el chirriar de la radiación. Puede distinguirse incluso la diferencia de energía en cada una de las partículas.

Tarjeta

Para el proyecto se utiliza una tarjeta con dos secciones (**figura 2**). La tarjeta del sensor puede separarse y se conecta con la del contador mediante un cable de tres líneas. Así tendremos la posibilidad de introducir el sensor en una carcasa a salvo de la luz.

El LCD y el pulsador se encuentran en la parte posterior, y el resto de componentes en la anterior. En cuanto al sensor de radiación D1, tenemos la posibilidad de elegir dónde lo colocamos después.

Probemos primero el circuito sin el diodo sensor. El operacional ahora mostrará un valor medio de continua a la salida. La entrada debería ser tan sensible que con sólo acercar un dedo, el contador ya tendría que reconocer una señal.

El sensor también puede colocarse en la parte trasera, como puede verse en la **figura 3**. En tal caso, la parte sensible del medidor sería el reverso de la tarjeta. Es importante que el fotodiodo BPW34 esté a salvo de la luz, y el área que lo rodea debidamente apantallada. En el montaje hemos de colocar debajo un poco de cinta aislante negra para evitar que posteriormente pueda pasar luz a través de la tarjeta verde-semitransparente. En la **figura 4** puede verse la cara de los componentes de la tarjeta prototipo.

La tarjeta ha de cubrirse en ambas caras en las zonas cerca del fotodiodo con papel de aluminio, posteriormente conectado a masa. Sólo así se logra un buen apantallado ante la luz y las interferencias, que pueden derivar en falsas señales. Bajo el papel de aluminio conviene utilizar nuevamente, para asegurarnos de que no haya cortocircuitos. Como contacto con el papel de aluminio podemos utilizar, por ejemplo, un

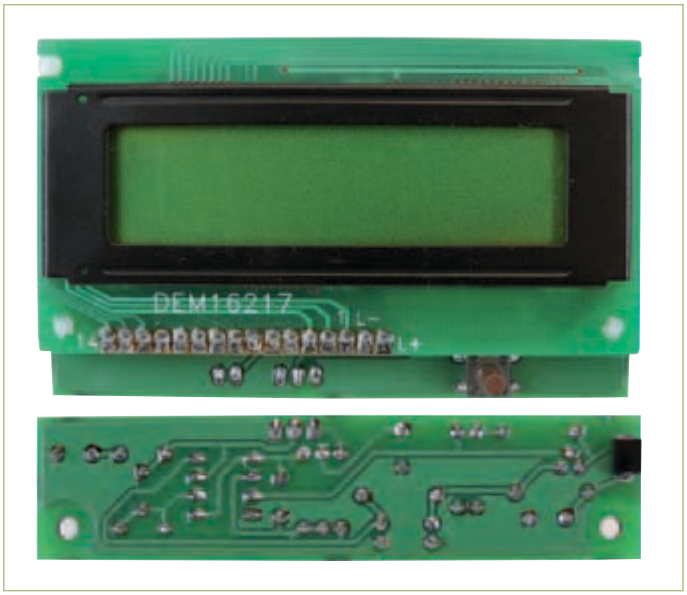


Figura 3. Montaje en la cara posterior de la tarjeta.

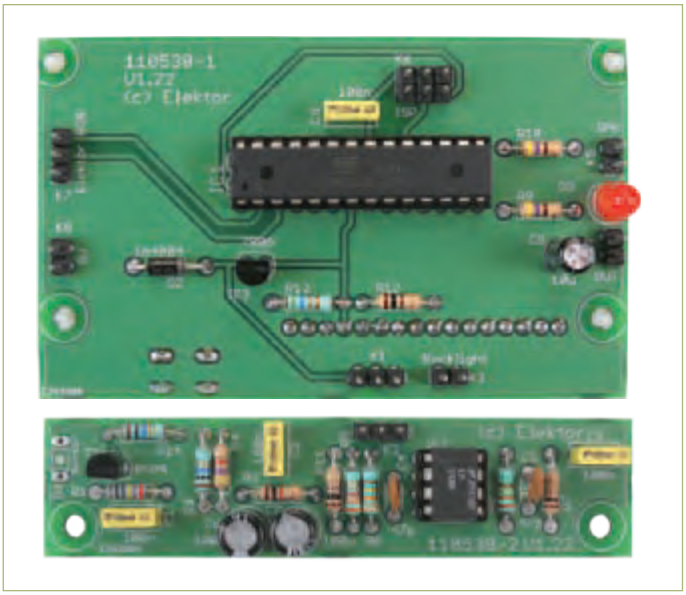


Figura 4. Cara anterior con los componentes.

tornillo y dos arandelas. El papel de aluminio ha de estar muy cerca del fotodiodo. Si esta distancia es demasiado grande, formaremos involuntariamente un micrófono de condensadores, y el contador podría reaccionar emitiendo un ruidoso silbido.
Ahora probemos el funcionamiento de la tarjeta con el fotodiodo aislado de la luz. En la salida aparecerá una tensión continua de unos 2 a 3 V. Podemos comprobar que el circuito está en realidad debidamente a salvo de la luz. Si el punto de trabajo se ha desplazado hacia arriba, significa que ha penetrado la luz. Si todo está en orden, el osciloscopio sólo mostrará ruido eventual de un orden de magnitud en torno a los 5 mV_{SS}.
Ahora ya podemos probar el dispositivo con una muestra radiactiva. Pongamos por ejemplo un mineral radiactivo directamente en el sensor. Los rayos gamma emitidos por la probeta generarán señales que se puedan oír claramente. Cada pulso mayor a cierto nivel se contabiliza. Este nivel de disparo puede ajustarse posteriormente en el software. Si no tenemos a mano una probeta radiactiva, nos tocará esperar. Finalmente, tras unos cuantos minutos una partícula procedente de la radiación cósmica chocará con nuestro sensor y será contabilizada (**figuras 5, 6 y 7**).

Medir la radiación con un asequible fotodiodo

Medidas con alfa

El BPW34 utiliza una cubierta plástica de un grosor capaz de ser atravesado por la radiación alfa. En lugar de un BPW34 también puede utilizarse un BPX61 al que le hayamos quitado la ventana de cristal. Entonces, el verdadero fotodiodo estará completamente libre y será sensible a las partículas alfa. En comparación con los rayos gamma, éstas producen señales cuyo tamaño es unas diez veces mayor. Puede utilizarse el mismo amplificador de instrumentación, pero ahora ha de colocarse independientemente, en su propio encapsulado a salvo de la luz y debidamente apantallado. La probeta a testear ha de estar en una habitación oscura, ya que incluso el papel de aluminio es demasiado grueso para la radiación alfa (véase la **figura 8**).

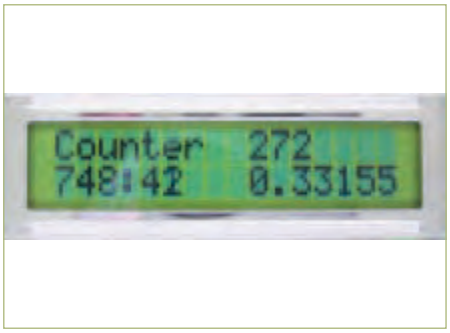


Figura 5. Medida del cero: 0,33 pulsos/minuto es un valor normal.

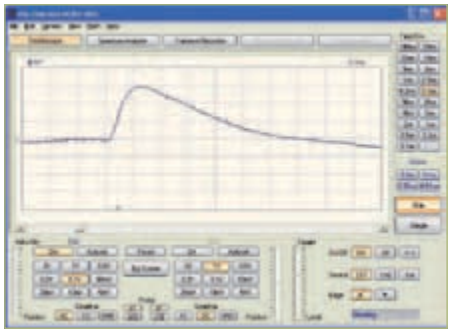


Figura 6. Pulso de medida independiente.

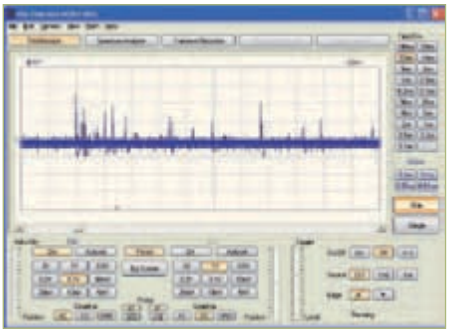


Figura 7. Ruido de fondo y señales útiles.

Código 1:
valor medio y nivel de disparo

```
Readeeprom L , 1
 If L = 255 Then L = 10
 U = 0
 For N = 1 To 1000
   D = Getadc(0)
   U = U + D
 Next N
 U = U / 1000
 Um = U
 U0 = Um + L
 N = 0
```

Código 2:
detección de un pulso

```
Do
   Max = U0
   Do
     D = Getadc(0)
   Loop Until D > U0
   Portb.0 = 1
   Portb.1 = 1
   If D > Max Then Max = D
   Do
     D = Getadc(0)
     If D > Max Then Max = D
   Loop Until D < U0
   N = N + 1
   Max = Max - Um
   If Max > 255 Then Max =
255
   Print Chr(max);
   Portb.0 = 0
   Portb.1 = 0
 Loop
```

Código 3:
medida del tiempo y muestra de datos en el LCD

```
Tim1_isr:
  Timer1 = -7812
  S = S + 1
  If S = 60 Then
    S = 0
    M = M + 1
    Count = N
    Count = Count / M
    Locate 2 , 10
    Lcd Count
    Lcd „     „
  End If
  Locate 1 , 10
  Lcd N
  Locate 2 , 1
  Lcd M
  Lcd „:"
  Lcd S
  Lcd „  „
```

Código 4:
tratamiento de los valores de energía transferidos

```
Private Sub Timer1_Timer()
  While INBUFFER() > 0
    d = READBYTE()
    bin(d) = bin(d) + 1
  Wend
  For n = 1 To 255
    x1 = 2 * n
    x2 = 2 * n + 2
    y1 = 200 - bin(n)
    y2 = 200 - bin(n + 1)
    If y1 > 255 Then y1 = 255
    If y2 > 255 Then y2 = 255
    Picture1.Line (x1, y1)-
(x2, y2)
  Next n
End Sub
```

Código 5:
ajuste del umbral de disparo

```
Private Sub Command2_Click()
  l = HScroll1.Value
  SENDBYTE l
End Sub

Private Sub Command4_Click()
  l = 100 + HScroll1.Value
  SENDBYTE l
End Sub
```

Para retirar la cubierta de cristal del BPX61 hemos de utilizar una pulidora muy rápida y precisa (como la Dremel). Hemos de ir con mucho cuidado ya que dañar la estructura o las patillas de conexión supondría el fin del diodo.
Como apantallado sirve una sencilla lata. Ésta ha estar conectada siempre a masa, para que actúe en todo momento de apantallado eléctrico. Una vez colocada esta cubierta podemos comenzar con las medidas.
En el osciloscopio pueden verse las señales alfa con picos de hasta 2 V. Sin embargo, al mismo tiempo también pueden distinguirse señales más débiles, ya que el BPX61, al igual que el BPW34, es sensible a los rayos gamma. Por lo tanto, el tipo de radiación puede deducirse según la amplitud de los pulsos.

Figura 8. Apantallado.

Firmware

El firmware (que puede descargarse gratuitamente en [2]) está codificado en Bascom-AVR y es relativamente simple y fácil de entender. La tensión de salida del preamplificador en reposo es de unos 2 V. En funcionamiento, adicionalmente tenemos los pulsos del sensor. En realidad, para que un contador pueda utilizar estos pulsos, hace falta un comparador. Pero el ATmega ya es lo suficientemente rápido como para hacer esta tarea de forma independiente. Aparte, en el inicio se ejecuta un balance de cero midiendo más de 1000 puntos (véase el **código 1**). El valor medio U se eleva mediante el nivel de disparo L, para establecer una gran diferencia respecto a los ruidos y posteriormente servir U_0 como valor comparativo para el procesamiento de los pulsos.
Durante las auténticas mediciones (**código 2**), el comparador por software también controla ambas salidas digitales PortB.0 y PortB.1. En B.0 está conectado un LED que parpadea brevemente con cada pulso reconocido. En B.1 puede conectarse un mini-altavoz con una resistencia en serie (y opcionalmente hasta un potenciómetro de volumen).
Adicionalmente se busca el máximo de cada pulso y se envía en un byte al PC mediante el puerto serie. Sólo en un byte para que no se provoquen retardos. Esto también significa que la amplitud de los pulsos está limitada a un valor de 255, lo cual equivale a

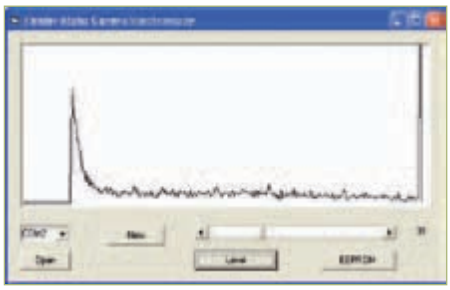


Figura 9. Espectro alfa de una muestra de uraninita.

Figura 10. Radiación gamma: uraninita tras papel de aluminio.

Figura 11. Medición en beta con cloruro de potasio.

1,25 V. Pueden darse pulsos mayores, pero se enviarán como 255.
El display muestra continuamente el estado de la cuenta, y éste se actualiza una vez por segundo (**código 3**). Aparte, el tiempo de medida se indica en la línea de abajo según minutos:segundos, desde el comienzo de las mediciones. Además, el programa calcula cada minuto la media de impulsos por minuto.

Midiendo la radiación de fondo

La sensibilidad del medidor depende en gran medida del umbral de respuesta del comparador. Éste está fijado por defecto a 10, pero puede cambiarse posteriormente mediante el puerto. Un umbral de tres etapas A/D es óptimo para distinguir los pulsos del resto del ruido. Para cambiar el umbral sólo hay que enviar un único byte al dispositivo. Los valores de hasta 100 se toman automáticamente como nuevo umbral de respuesta. Si queremos almacenar un nuevo umbral como ajuste por defecto en la EEPROM del controlador, añadimos 100. Un byte 103 no tendrá un efecto inmediato, sino que fija definitivamente el umbral a partir del reinicio en 3.

PC-Software

El programa en VB AlphaGamma (descarga gratuita en [2]) recibe todos los bytes entrantes y los ordena en sus 255 posiciones (bins). Tras algún tiempo veremos en qué estado energético se miden especialmente más resultados. El espectro de energía se representa de forma sencilla en el diagrama (**figura 9**). Las altas energías pertenecen principalmente a partículas alfa. Debido a que tenemos una probeta especialmente gruesa, las partículas alfa pierden en proporción un montón de energía en su camino al sensor. Por ello, con esta muestra no se aprecian líneas rugosas. En medidas gamma puras con el BPW34 la franja superior permanecerá vacía. Las partículas alfa ofrecen mucha más energía que la especificada por el umbral, sencillamente los picos resumen el máximo valor.
El **código 4** muestra la verdadera rutina del timer. Aquí se leen y analizan todos los bytes, que se encuentran en el buffer. Mediante los valores leídos, se llena el array bin(255) y se representa gráficamente.
El programa permite al mismo tiempo ajustar el umbral de respuesta en un rango de 2 a 100. Opcionalmente la configuración puede hacerse directamente o ser almacenada en la EEPROM (**código 5**).
Las partículas alfa pueden registrarse fácilmente. Con el BPX61 abierto, basta con una cubierta de papel de aluminio. El espectro de la uraninita claramente se desplaza hacia abajo (**figura 10**).
Las partículas beta también dan origen a señales cuantificables, pero sus amplitudes son similares a las de los rayos gamma y por lo tanto difíciles de distinguir. Como prueba para la sensibilidad ante partículas beta, se ha hecho una medida de larga duración con el BPX61 y una pequeña muestra de cloruro de potasio. Dicha muestra contiene potasio 40, que es radiactivo. El 90 % de las desintegraciones desprenden partículas beta con una energía máxima de aproximadamente 1,3 MeV. El 10 % restante generan cuantos gamma de 1,5 MeV. El espectro beta muestra una división energética muy característica, con una clara máxima energía. El espectro gamma consiste en una línea rugosa. El espectro total (**figura 11**) tiene el patrón previsto. Éste ilustra cómo el fotodiodo es capaz de detectar principalmente radiación alfa, beta y gamma.

(110538)

Enlaces:

[1] www.elektor.es/110372
[2] www.elektor.es/110538

Kit de montaje

Para este proyecto, dispone de un kit de montaje que incluye la tarjeta y todos los componentes (incluido el controlador programado). También puede encargarse adicionalmente un display. El precio y el resto de información ampliada puede encontrarse en la web de este artículo [2] y en las páginas de la tienda, al fina de esta edición.

SPICE it up
Simular con LTspice

Raymond Vermeulen (Laboratorio de Elektor)

La simulación de circuitos es útil para todo aquel que se dedique a la electrónica, desde estudiantes y aficionados hasta los profesionales. De esta forma se pueden verificar mediciones rápidamente y hacer adaptaciones al diseño sin tener que hacer un montón de cálculos, y también se pueden ver las características no-ideales que se descuidan en las fórmulas, etc. En este artículo damos brevemente el primer empujoncito a aquellos que aún no están familiarizados con los simuladores basados en SPICE.

Para este curso de introducción hemos elegido el LTspice, un programa de simulación de Linear Technology basado en SPICE. Se puede adquirir gratuitamente (sin registrarse) y es fácil de usar. Una desventaja es que el programa viene de forma estándar con las librerías de componentes que, por supuesto, contienen productos de LT. No hay componentes de otros fabricantes, pero el usuario sí los puede añadir. El programa es muy adecuado para la simulación de circuitos construidos con productos de LT y para esto el sitio Web de LT dispone de un montón de ejemplos de simulación. Por supuesto que para una funcionalidad más amplia y obtener más librerías puedes optar por un programa comercial como es el Micro-Cap ó Orcad Pspice, pero estos programas no son baratos. Experimentar con LTspice tiene al menos una ventaja: si conoces un programa SPICE, puedes manejar el resto.

El origen

SPICE fue originalmente desarrollado en los años 70 por la universidad de Berkeley para analizar la dureza de radiación en circuitos electrónicos, por encargo del Ministerio de Defensa de EEUU. En los años siguientes las posibilidades se fueron ampliando, como son la simulación de componentes más complejos y más funciones de simulación. Después de un tiempo, aparecieron las variantes comerciales que utilizaban frecuentemente SPICE como base y construyeron su propia capa (interfaz de usuario) alrededor.

Empezar con LTspice

Accede al sitio Web http://www.linear.com, haz clic sobre 'Design support' en la parte superior, a continuación en 'Design Simulation'. Desde esta página puedes descargar LTspice.
Ejecuta el fichero descargado e instala el programa en un sitio de tu elección. Los usuarios de Linux pueden utilizar este programa en combinación con Wine. Esto se ha probado bajo 'Ubuntu 11.04 Natty Narwhal'.
A continuación ejecuta el programa.
Crea una nueva hoja para el esquema pinchando en el icono situado en la parte más a la izquierda:

Figura 1. La elección de los componentes.

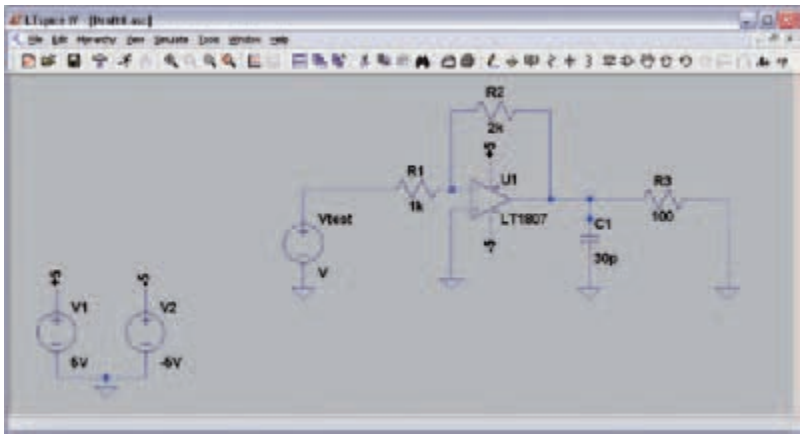
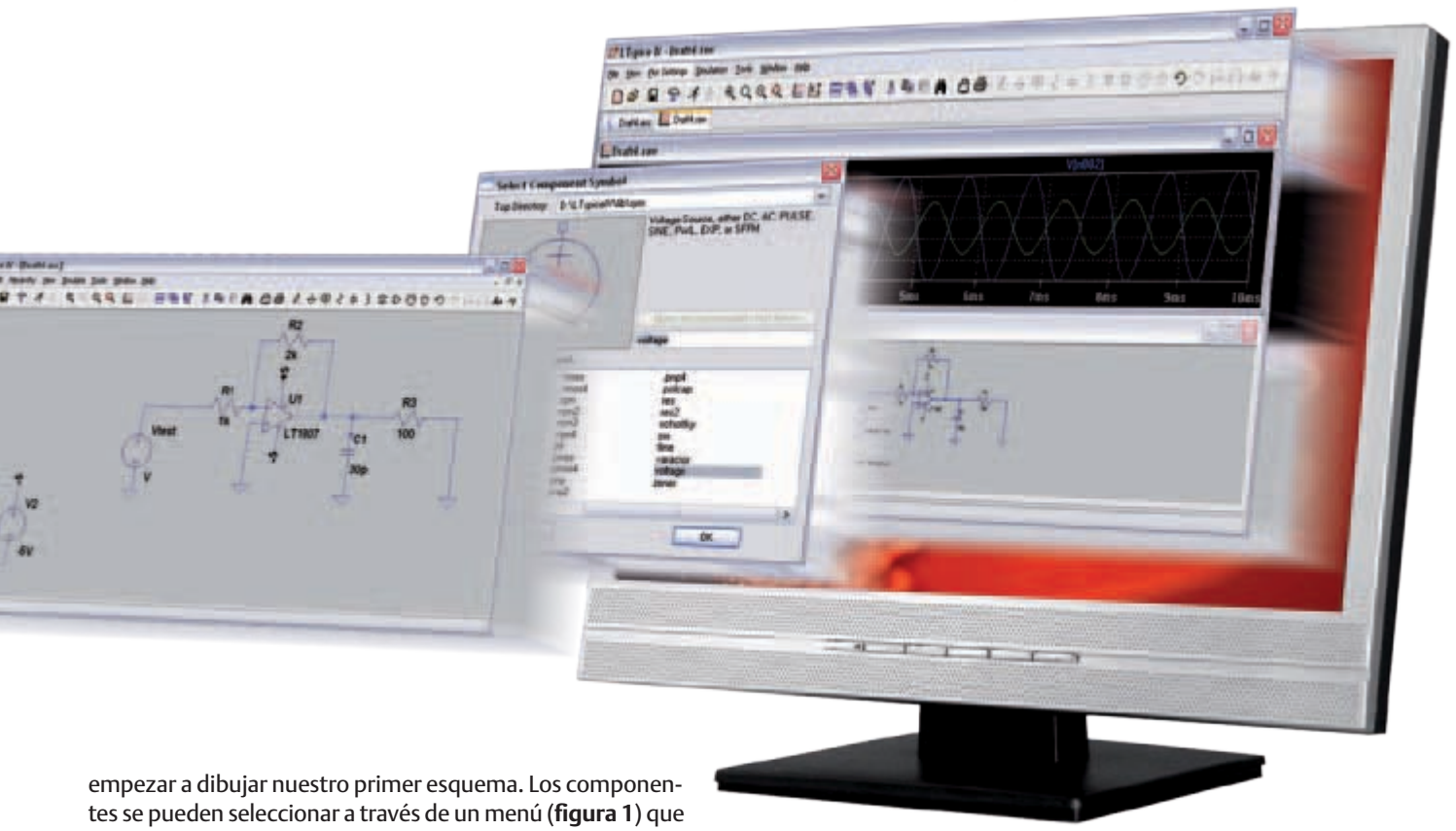


Figura 2. Nuestro primer esquema.



empezar a dibujar nuestro primer esquema. Los componentes se pueden seleccionar a través de un menú (**figura 1**) que aparece cuando pinchas en el botón 'component':

Los componentes estándares, como son los puntos de masa, resistencias bobinas, condensadores y diodos, se pueden seleccionar a través de la barra del menú:

Con los botones aquí marcados puedes construir el esquema a tu gusto:

Es aconsejable jugar un poco con ello para tener una idea del funcionamiento.
Hay combinaciones de teclas ('hotkeys') muy útiles: 'ctrl+R' para girar un componente y 'crtl+E' para hacer el efecto espejo.
En la **figura 2** puedes ver el primer esquema que hemos dibujado.
Ahora lo vamos a simular. Pero aún quedan dos cosas que tenemos que configurar: la fuente de tensión y los parámetros de simulación.
Si haces clic con el botón derecho sobre Vtest, aparece un menú donde puedes configurar las características básicas de la fuente de tensión (tensión DC y resistencia interna). Pero en este ejem-

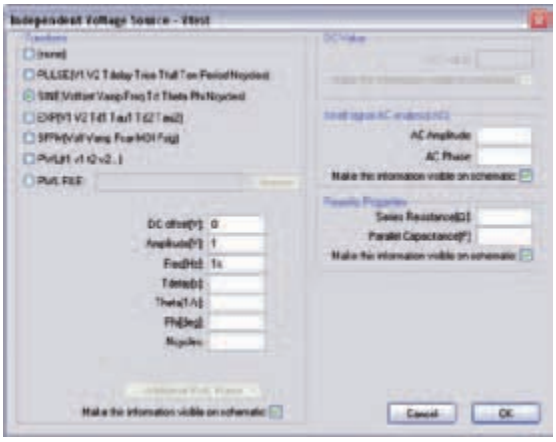


Figura 3. Las configuraciones avanzadas de la fuente de tensión.

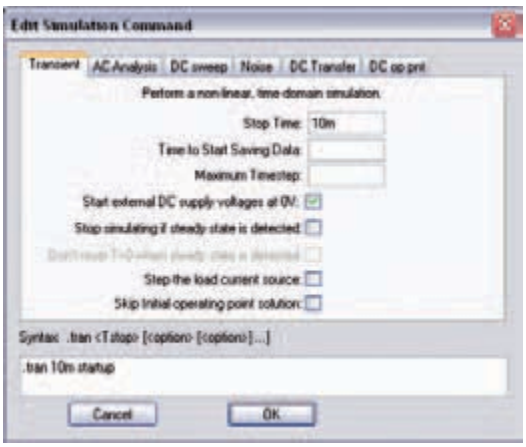


Figura 4. Opciones de configuración para la simulación.

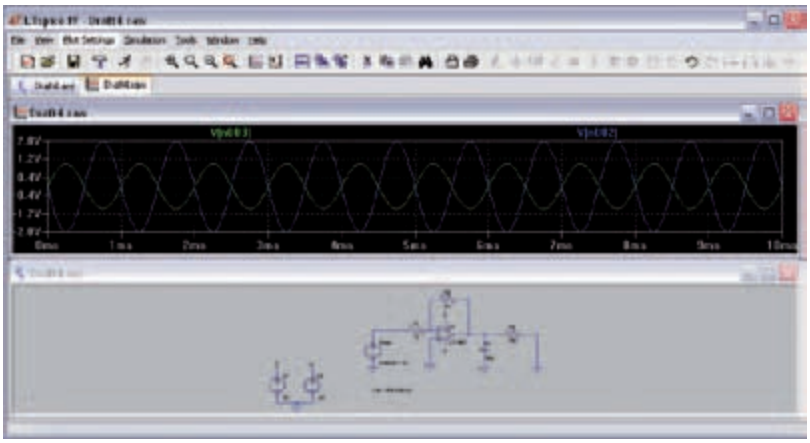


Figura 5. Las señales simuladas con el esquema que hay abajo.

plo queremos simular el comportamiento del circuito ante una tensión alterna. Para eso suministramos un seno de 1 KHz con una amplitud de 1 V y un offset de 0 V. Haz clic luego en 'Advanced'. Entonces aparece el menú de la **figura 3**. En él se pueden configurar varias formas diferentes de señales. Aquí hemos seleccionado el botón radio al lado de 'SINE' y hemos introducido los valores deseados.

Ahora vamos a simular el circuito en el dominio de tiempo para lo cual hemos de adaptar nuestras configuraciones de simulación. Esto se hace a través de 'Edit Simulation Cmd':

Figura 6. Selección de un análisis FFT.

Figura 7. Configuración y selección del punto nodal para el FFT.

Consejos

Componentes estándar

A la hora de dibujar un esquema, notarás que en la selección de algunos componentes aparece una larga lista de tipos en la ventana de selección de la figura 1, pero con otros componentes no aparece ninguna. Haz clic, por ejemplo, en [Opamps], entonces aparecerá una larga lista de (principalmente) tipos de LT, en cambio si seleccionas un transistor NPN no aparece ninguna. Sin embargo sí existe. Para eso hay que colocar primero un transistor estándar en el dibujo y luego hacer clic con el botón de la derecha. Selecciona 'Pick new transistor'. Luego aparecerá un amplio listado con modelos corrientes de varios fabricantes. Sucede lo mismo con los demás componentes estándares.

Componentes adicionales

Lógicamente el número de componentes en LTspice no es infinitamente grande. Puedes encontrar muchos componentes adicionales en el grupo de usuarios 'LTspice' de Yahoo:

http://tech.groups.yahoo.com/group/LTspice/

Entonces aparece la ventana de la **figura 4**. Como queremos ver el curso de la señal de salida en el tiempo, seleccionamos la pestaña 'transient'. Aquí podemos modificar los parámetros de la simulación. En este caso hemos optado por que la simulación dure 10 ms y que las fuentes de tensión continua empiecen en 0 V. Cliquea sobre 'OK' y coloca en un sitio que elijas del esquema, el recuadrado con la información de configuración que aparece donde se encuentra en ese momento el cursor del ratón. (Para ver un análisis del dominio de frecuencia habrá que abrir a continuación la pestaña 'AC analysis' y configurarla.)
Ya puede empezar la simulación. Para eso haz clic sobre el botón 'RUN':

Aparece una nueva ventana donde se muestran las señales simuladas. Haz clic en los puntos de medición deseados de la parte aún visible del esquema que se encuentra debajo. En este ejemplo hemos elegido la salida de la fuente de prueba y la salida del operacional. Las señales presentes en estos puntos aparecerán en la ventana de señales (**figura 5**).
Si haces clic luego con el botón derecho sobre la forma de la onda, aparecerá un menú con opciones (**figura 6**). Ahí puedes modificar muchas características de la representación de las señales simuladas. Puedes optar, entre otras, por una FFT o por una representación del espectro de frecuencias. Haz clic para eso sobre 'FFT' por debajo de 'View'. Entonces aparece una ventana donde se puede ajustar todo tipo de configuraciones FFT (**figura 7**). Selecciona el punto nodal en la parte superior sobre el que queremos ejecutar el análisis FFT. En este caso es la salida del operacional. La **figura 8** muestra el espectro de la señal.

Ahora podemos empezar a descubrir todas las posibilidades del software. Prueba también otros programas de simulación, existen demos de prueba o para estudiantes de muchas versiones con las que puedes adquirir experiencia. A menudo el manejo es diferente, pero las cosas importantes que hay que hacer son iguales: Dibujar el esquema, configurar una o varias fuentes de pruebas, configurar la simulación y estudiar la salida.

(110543)

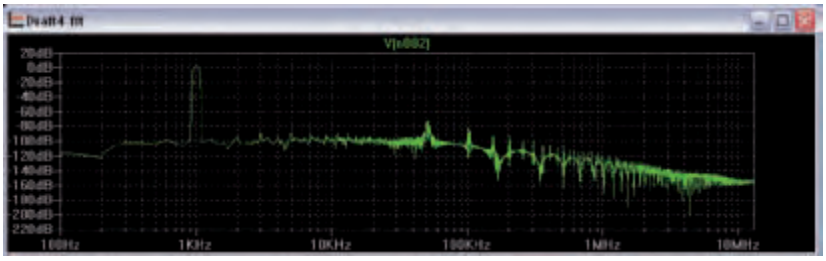


Figura 8. El espectro FTT calculado.

¡Que viene el bus! (9)

Pasemos a nuestro propio control

Quien haya querido probar su propia aplicación del bus, habrá tenido que hacer algunos cambios en una de nuestras demos, o bien programar todo desde cero. Ahora esto ha cambiado: podemos diseñar y controles del bus e interfaces de usuario personalizados en tiempo récord. Ya que el concepto se basa en HTML y Javascript, esta misma central puede utilizarse en distintas plataformas, tanto PCs como smartphones.

ElektorBusBrowser en un smartphone con Android.

Jens Nickel

La domótica y otras aplicaciones en el campo de la medida, control y regulación suelen servirse de un centro neurálgico con display, desde el que se pueden mostrar los valores o cambiar la configuración del usuario. Esta central puede ser por ejemplo un PC, programado en VisualBasic. Pero también un smartphone o un tablet-PC, en el que se ejecute el sistema operativo Open-Source Android, sirve como central de control perfectamente. Podemos enlazar nuestras propias aplicaciones en lenguaje Java con el eficiente entorno de Android, aunque para los principiantes tenga una curva de aprendizaje bastante pronunciada.
Quien pueda manejarse en dicho lenguaje, con el entorno y por lo tanto con las herramientas de desarrollo correspondientes, naturalmente también podrá implementar los protocolos del ElektorBus con su propia aplicación de control. Si empezamos a mezclar el código de los protocolos y la verdadera aplicación (igual que ha ocurrido con el software demo presentado hasta ahora), los cambios y las ampliaciones resultan difíciles de implementar. Y si cambiamos de plataforma/dispositivo, tendremos que empezar de cero.
Por ello, necesitamos de una librería que

- Incluya los protocolos del ElektorBus, de forma que el desarrollador pueda centrarse en su propia aplicación del bus.
- Separe con claridad el código de la aplicación y del protocolo.
- Permita una programación y distribución sencillas, que puedan manejar muchos aficionados a la electrónica.
- Que sea independiente de la plataforma, de modo que la misma aplicación pueda ejecutarse tanto en un PC como en un Smartphone.

¿Que no existe? Bueno, ¡aquí viene!

El HTML se entiende

Antes de que expliquemos cómo se utiliza la librería, queremos centrarnos en cómo funciona el método al completo. De un primer vistazo el concepto puede parecer algo complejo e incluso extraño. Pero en la práctica, las ventajas son impresionantes en comparación con la programación tradicional, con lo que la idea puede ser útil tam-

Productos y servicios Elektor

- Nodos experimentales (tarjeta 110258-1 o tercer set de tarjetas 110258-1C3)
- Conversor USB/RS485 (montado y probado 110258-91)
- Descarga gratuita del software (firmware del controlador más software de PC)
- Todos los productos y descargas están disponibles en la web de este artículo: www.elektor.es/110517

bién para otros proyectos de Elektor que requieran control mediante un PC. Aparte, se utilizan sistemas similares en los desarrollos de software modernos, como por ejemplo las “apps” móviles. Por ello, recomendamos a los principiantes que dediquen algo de tiempo a pelearse con las siguientes secciones.

Primero está la independencia de la plataforma: esto lo conseguimos ya que la auténtica aplicación del bus está programada con un entorno de desarrollo basado en HTML y Javascript. Este dueto es una auténtica panacea, pueden representarse UIs (en un navegador) en PCs Windows, Mac, Linux y diversos dispositivos móviles. Otra de las ventajas es que al estar basado en formularios HTML puede accederse desde Internet con las fascinantes aplicaciones que esto conlleva, en cuanto al control remoto. Aparte, el HTML tiene un potencial creciente: el HTML5 añade muchas características, como por ejemplo las bases de datos locales, los gráficos en 3D y demás.

Navegador especial

Lógicamente, los entornos HTML creados para nuestro bus pueden visualizarse en un navegador web “normal” como Firefox o Internet Explorer, pero en cuanto a seguridad, estos navegadores pueden hacer bastante menos de lo que una aplicación cualquiera podría necesitar. Un navegador estándar no está diseñado, por ejemplo, para recibir o enviar datos por el puerto serie. Por ello, hemos desarrollado un navegador especial, que aparte de poder representar formularios HTML también es apto para el ElektorBus. Ya que el *ElektorBusBrowser* accede al USB y a otras funciones de dispositivos, cada plataforma ha de tener el propio. No obstante, esto no supone un inconveniente, ya que normalmente el usuario no tendrá que modificar en absoluto el código fuente. Por ello, también podremos a disposición el *ElektorBusBrowser* como aplicación ya terminada (por ejemplo en .exe si se trata de un PC). Si cambiamos de plataforma, sólo tendremos que instalar nuevamente el *ElektorBusBrowser* correspondiente y los archivos HTML/Javascript de nuestra propia aplicación en la carpeta correcta. ¡Y listo!

La captura de pantalla de la **figura 1** muestra el primer *ElektorBusBrowser* realizado: como un navegador normal, dispone una ventana que representa el contenido HTML de nuestra propia aplicación, que es la que más espacio ocupa. HTML y Javascript forman por así decirlo la estructura básica, y en un browser programado en VB.NET o Android-Java supondría el exterior de la aplicación (véase la **figura 2**). También podemos llamar *Host* (“anfitrión”) al *ElektorBusBrowser*.

Librería de protocolos

Este *Host* aceptará un *ElektorBusMessage* mediante el puerto serie del correspondiente dispositivo. Posteriormente, el mensaje se procesará según los protocolos descritos en adelante; los datos útiles se pasan a la verdadera aplicación de control (en HTML y Javascript). Si el usuario pulsa un botón en el entorno HTML, Javascript generará un mensaje con los datos útiles correspondientes (un ejemplo de la última entrega sería establecer el valor límite de un sensor). Ahora la información es procesada “hacia delante” por el *ElektorBusBrowser*, que envía el mensaje.

En principio pueden implementarse los tres protocolos a la vez en Host, el *ElektorMessageProtocol*, el *HybridMode* (opcional) y el *ApplicationProtocol*. Por otra parte, también resulta posible analizar los 16 bytes “en bruto” una vez “dentro” y ejecutar los protocolos directamente con Javascript. Sin embargo, he aquí la forma mixta: el sencillo *ElektorMessageProtocol* y el algo crítico (en cuanto al tiempo) *HybridMode*, junto con el planificador, se manejan dentro del código del host; el *ApplicationProtocol*, que requiere algo más de código y probablemente sea ampliado por muchos lectores, se implementa mediante una pequeña librería de Javascript (**figura 3**). Y de esa manera, detalladamente funciona así: El *host* recibe los 16 bytes mediante un *ElektorBusMessage* por el bus, con ayuda de la sincronización del byte de start descrita en [1]. Posteriormente el mensaje se “desempaqueta”, y se obtiene la estructura de datos con (entre otros) la dirección del emisor, la dirección del receptor y los ocho bytes de datos útiles. Estas partes individuales se codifican en un solo string (*InCommand*) y son gestionadas en

Figura 1. Captura de pantalla del primer ElektorBusBrowser, con una aplicación de ejemplo ejecutándose.

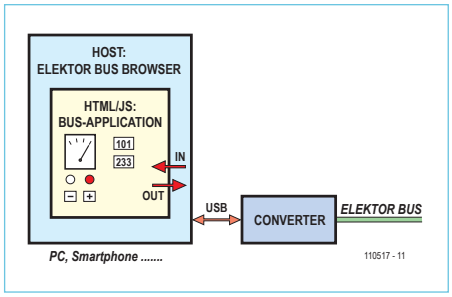


Figura 2. El Host dependiente de la plataforma es sólo para las funciones básicas (puerto serie, sincronización). La verdadera aplicación del bus se ha implementado HTML/Javascript.

adelante por Javascript (véase la **figura 4**). InCommand (ver cuadro) es texto plano, y garantiza la compatibilidad con distintas plataformas.

Javascript se sirve de InCommand y convierte el string en una sencilla estructura de datos de nombre *Message*, que ahora ya puede descodificarse. Según el *ElektorApplicationProtocol* también tenemos las llamadas *Parts*, que podemos traducir como unidades de información (ver el cuadro “mensajes y partes”). De este modo, una unidad puede constar, por ejemplo, de un valor transferido de 2 bytes (de -1023 a

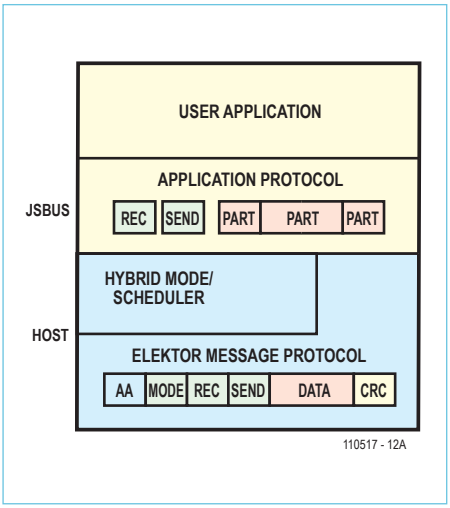


Figura 3. Stack de protocolo estándar para el ElektorBus (HybridMode y planificador opcionales). En el ElektorMessageProtocol el responsable es el Host, para la ApplicationProtocol, la librería en Javascript sobre JSBus.

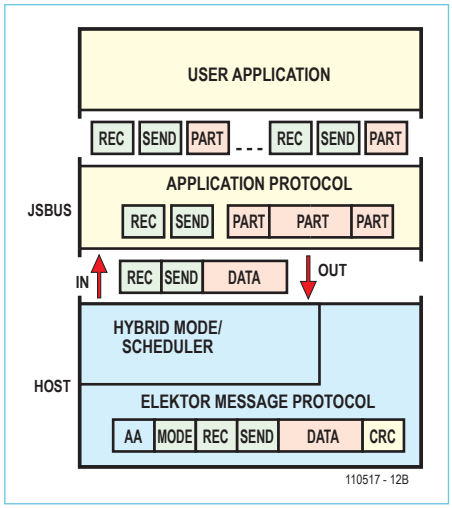


Figura 4. El Host y la librería JSBus intercambian información sobre las confirmaciones recibidas y los mensajes enviados (en formato de texto).

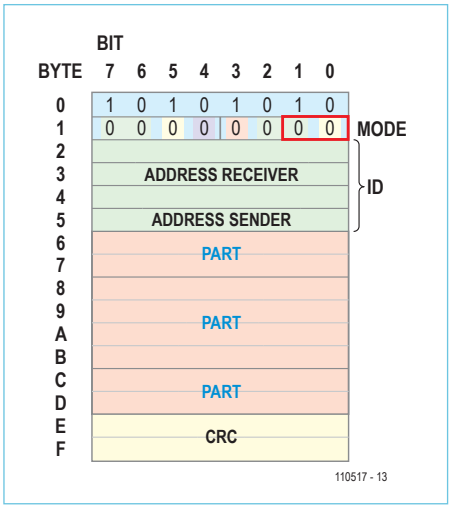


Figura 5. El ApplicationProtocol permite transferir a la vez multitud de cifras o unidades de información, gracias a los 8 bytes de datos (por ejemplo, los valores límite de las alarmas). El diagrama muestra con claridad las tres partes descritas: dos veces con dos bytes y una vez para tamaños de 4 bytes.

1023) o incluir el mensaje "¡se ha superado el valor límite de Sensor2!". Ya que podemos transferir muchas de estas pequeñas unidades de información en un solo mensaje (**figura 5**), la librería en Javascript para un *Message* dispone de múltiples *Parts* (un array de partes). Posteriormente, las partes recibidas ejecutarán funciones en Javascript, que el usuario puede llenar con su propio código. Posteriormente, dicho código de usuario transferirá a un cuadro de texto en HTML el valor numérico que se transmitió con la parte.

Tipos de comandos

Al enviar todo va en la otra dirección. Al hacer clic en un botón en HTML se llama a una función en Javascript escrita por el usuario, que genera una (o varias) partes. Las partes que han sido identificadas para el mismo receptor se codifican como un solo mensaje. El objeto del mensaje es traspasado mediante *OutCommands* (otra vez es sólo texto plano) en Javascript hacia fuera hasta el *ElektorBusBrowser*. Éste genera 16 bytes que se envían a través del bus. Tras completar el trabajo, el *Host* devuelve

InCommand y OutCommand

La aplicación del bus codificada en Javascript y el *ElektorBusBrowser* (codificado por ejemplo en VB.NET) como Host se comunican entre sí mediante una simple cadena de caracteres. Gracias a la llamada sintaxis JSON se codifica una estructura de datos con información, que Javascript ha de transferir al Host (hacia afuera) o el Host (hacia dentro) a la aplicación en Javascript. La estructura de datos de las instrucciones In y OutCommands es bastante parecida:

OutCommand:

Command	Tipo de comando ("Send", "Url", "Scheduler")
Url	Nombre de archivo de la página HTML (solo "Url")
Options	Reservado para su uso posterior
Mode	Byte de modo del mensaje enviado (importante sistema de Acknowledge)
Receiver	Dirección del receptor
Sender	Dirección del emisor
Data	Array de ocho bytes de datos (o las direcciones de hasta ocho ScheduledNodes)

InCommand:

Command	Tipo de comando ("Rec", "Status")
Mode	Byte de modo del mensaje recibido (o sea, Status 2 = ok, -1 = Error)
Valid	¿Checksum ok? (aún sin implementar)
Receiver	Dirección del receptor
Sender	Dirección del emisor
Data	Array de ocho bytes de datos

Un InCommand en sintaxis JSON tiene el siguiente formato:

```
{"Command":"Rec","Mode":0,"Valid":0,"Sender":
2,"Receiver":10,"Data":[0,0,64,1,0,0,0,0]}
```

En nuestro primer *ElektorBusBrowser* se muestran los In y OutCommands por motivos de debugging (ver la figura 1 más abajo).

un acuse de recibo en Javascript. Para ello también se utiliza un InCommand, que es del tipo “Status”.
Existen otros tipos de OutCommands (ver cuadro); con estos la aplicación en HTML/Javascript puede controlar al *Host*. El OutCommand “Url” obliga al *Host* a cargar una nueva página en HTML. De este modo podemos diseñar una aplicación que por ejemplo disponga de diferentes formularios, seleccionables mediante un menú. El OutCommand “Scheduler” activa o desactiva el planificador. Aparte, en el array de datos se da un listado de direcciones de hasta ocho nodos que han sido preguntados.

Aplicación de ejemplo

Lo mejor es utilizar un ejemplo para mostrar como funciona una librería de Javascript y el *ElektorBusBrowser*. Podemos servirnos del hardware de la anterior entrega en 1:1, y hacer solamente unos cambios mínimos en el firmware de ambos nodos (archivo en BASCOM disponible en [2]). La **figura 6** muestra nuestra unidad nuevamente: el nodo 2 equipado con una foto-resistencia transmite valores continuamente al master domótico. Desde el master puede cambiarse la unidad de medida de los valores, así como fijar el límite inferior. Si se sobrepasa dicho valor límite, el sensor avisará con un mensaje de alarma. Posteriormente el master envía al nodo 1 un mensaje para indicar que ha de abrirse el relé conectado. Aparte, el sensor confirma el mensaje de alarma.
Podemos descargar la primera versión del *ElektorBusBrowsers* para PC desde la web del artículo [2], así como la librería de Javascript JSBus.txt y la aplicación de ejemplo, que aparecerá a modo de dos páginas web llamadas “Index.htm” y “Limit.htm”. La carpeta UIBus, que incluye los tres archivos nombrados previamente, ha de colocarse en el escritorio. Tras iniciar el ElektorBusBrowser.exe se dirige al primer formulario HTML (ha de llamarse siempre “Index.htm”). Una vez conectado el puerto serie e iniciado el planificador, se mostrarán –como en el software demo para PC de la anterior entrega– los valores del foto-sensor en un cuadro de texto. Con los botones disponibles actualmente podemos selec-

Mensajes y partes

Dentro de la librería Javascript se trabaja con dos estructuras de datos, para poder describir los *Messages* o las Parts enviadas (las unidades de información transferidas como valores de 2 bytes, mensajes de alarma, dimensiones de medida, etc.).

El objeto **Message** consta esencialmente de algunos parámetros ya conocidos del ElektorBusMessage:

Mode	Byte de modo
Receiver	Dirección del receptor
Sender	Dirección del emisor
Data	Array de ocho bytes de datos
Valid	¿Checksum ok? (aún sin implementar)

Dentro de los ocho bytes de datos, si fuera necesario, en el *ApplicationProtocol* pueden transferirse hasta cuatro partes separadas. Una parte se caracteriza por los siguientes parámetros:

Valid	¿Checksum ok? (aún sin implementar)
Sender	Dirección del emisor
Receiver	Dirección del receptor
Channel	Número de canal
Setflag	¿Valor actual o fijado?
Ackflag	AcknowledgeMessage o mensaje original (flag al nivel de aplicación)
Mode	Byte de modo del mensaje (con flags de Acknowledge al nivel de mensaje)
Parttype	Tipo de la parte, en el que se definen las constantes: PARTTYPE_VALUE2, PARTTYPE_VALUE4, PARTTYPE_VALUEFLOAT, PARTTYPE_LIMIT, PARTTYPE_SCALE.
Numvalue	Valor numérico transferido (por ejemplo de -1023 a 1023 en PARTTYPE_VALUE2)
Limit	0 = todo ok, 1 = por debajo del límite, 2 = por encima del límite
Quantity	Magnitud física, de 0 a 127, véase [3])
Unit	Unidad (de 0 a 3, véase [3])
Scale	Potencias de diez para el escalado (de -15 a +15)
Interval	Reservado
Preset	Reservado
Options	Reservado

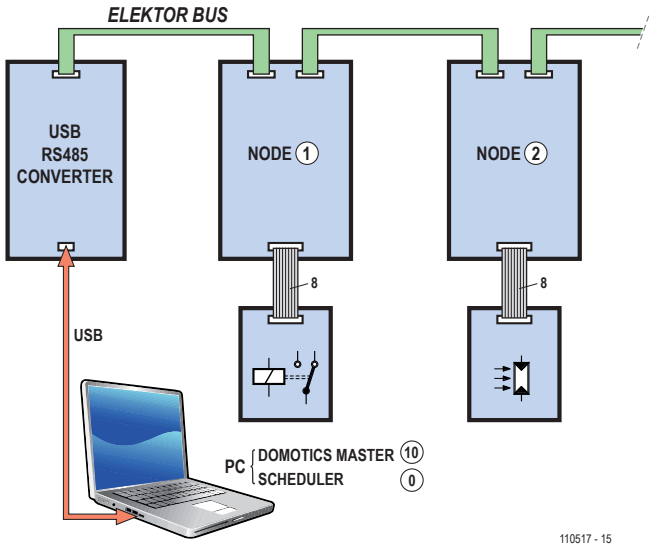


Figura 6. Para nuestro ejemplo utilizamos el hardware de la anterior entrega.

Figura 7. Supervisión de valores límite: todo es independiente de la plataforma y ha sido diseñado en HTML y Javascript.

cionar entre valores del ADC (de 0 a 1023) y Ohmios. La aplicación de alarma, por el contrario, se basa en un segundo formulario, con el fin de demostrar el uso de comandos "Url" (**figura 7**).

Hacia dentro

Tras este test, abrimos con un editor de texto el archivo "Index.htm", y así poder examinar su código fuente (ver **código**). Arriba del todo aparece una referencia a la librería de Javascript JSBus. Debajo (tras unos cuantos Script-Tags) le sigue el código en Javascript específico del usuario. La función ProcessPart(part) {...} es imprescindible; se la invoca cada vez que se recibe una parte en el JSBus. En la función, el desarrollador de la aplicación ha de especificar cómo procesar los valores recibidos, los mensajes de alarma y demás. En las características (Properties) de las partes, por ejemplo el emisor, se establece el canal y el valor a transferir mediante la sintaxis `part.característica`. Quien no esté familiarizado con la sintaxis de Javascript similar a C, aún así reconocerá fácilmente desde qué emisor se ha enviado cada parte correspondiente. Los valores de LEDs se transmiten al Channel1 (0 para apagarlo, 1 para encenderlo). Si la parte pertenece a Channel1, se activa un "radiobotón" en el formulario HTML. Esto se hace en la función `RadioButtonSetvalue`, implementada en nuestra librería JSBus (para las funciones más importantes ver el cuadro). Como parámetros esta función recibe la ID del radiobotón HTML así como un valor del 0 o 1 (OFF/ON). Ya que `part.Numvalue` en nuestro caso es únicamente o 0 o 1, podemos implementar esta expresión directamente en la llamada a la función. Como IDs para ambos radiobotones hemos elegido "LED1" y "LED2". De este modo podemos fijar los primeros parámetros de la función con la string "LED" y la dirección del emisor.

La función `SetSensorScale(quantity)` se llama mediante un clic en un botón HTML. Ahora ha de enviarse una parte que hará que el fotosensor transmita los próximos valores de medida como valores del ADC o en Ohmios. La estructura del código correspondiente es siempre la misma: con la línea `var parts = InitParts();` primero se inicializa con partes un array vacío. La siguiente línea es muy importante: añade otra parte al array existente. La función de librería `SetScale` hace esto de una forma bastante cómoda: como parámetro recibe el conjunto de partes a las que añadir un nuevo elemento, las direcciones del emisor y el receptor, el canal y byte de modo, así como los tres valores necesarios para el orden, la unidad y la escala en potencia de diez. Ésta devuelve el array de partes completo, y aquí podemos trabajar con más partes según cada función de Javascript correspondiente (ver cuadro).

Necesitamos ante todo el byte de modo, y que así podamos enviar un *AcknowledgeMessage* marcado a un nodo que esté enviando en una *FreeBusPhase* no segura (ver el cuadro "Un nuevo bit"). En el ejemplo el nodo 1 informa del estado del propio LED de test, cuando se pulsa el botón de test. Sin embargo, el envío del correspondiente mensaje de confirmación por parte del master no está implementado en el código de aplicación actual. Por ahora esto se hace automáticamente en la librería de Javascript (ver la función `ProcessReceivedParts(parts)` en JSBus.txt).

Hacia fuera

Mediante `SendParts(parts, overrideQueue);` se envían las partes generadas (en nuestro caso sólo una) una vez codificadas y preparadas. Aparte, la librería junta hasta cuatro partes en un array con el mismo emisor y receptor, y prepara automáticamente un único mensaje. Las partes que no quepan en un sólo mensaje se codifican en varios mensajes y se envían uno tras otro. El desarrollador de la aplicación tampoco tiene que preocuparse de ello. Los mensajes se disponen automáticamente en una *Queue* (cola); cuando el Javascript del Host da la orden, se envía el primer mensaje, luego el siguiente y así en adelante. Ahora podremos comprender el segundo parámetro `overrideQueue`: una nueva llamada a SendParts puede sobrescribir los mensajes no enviados o expirar si los mensajes ya se encuentran en la cola (en el archivo Limit.htm utilizamos estas dos posibilidades al activar o resetear las alarmas, teniendo en el master dos mensajes para enviar).

Las próximas dos líneas del código en Javascript pasan a ser un simple texto en nuestro formulario HTML. Para que esto funcione, el texto siempre ha de tener una ID ("unidad"). Dependiendo del valor de `quantity`, el texto se pasa a "Ohmios" o "valores del ADC". En la librería se han definido las medidas más importantes y unas cuantas constantes, `RESISTANCE` por ejemplo tiene el valor 18. Atención a los lectores conocedores de BASIC: ¡para comparar se escribe la doble igualdad y el código Jacascript distingue entre mayúsculas y minúsculas!

Y ahora respecto al HTML: consiste en una serie de elementos introducidos por tags como "DIV" o "INPUT". Son especialmente importantes para nosotros los elementos de INPUT y BUTTON. Con los elementos de INPUT podemos ver el tipo (Radio[button] o Text[box]) y la ID, y esto es lo que hace que los elementos de entrada puedan ser reconocidos posteriormente por Javascript. El texto que queramos cambiar durante el texto de ejecución es mejor marcarlo con los tags <DIV> o <SPAN>, siendo el primero exactamente igual para nuestros propios párrafos.

Código: el archivo Index.htm

```
<SCRIPT src='JSBus.txt' Language='javascript' ></SCRIPT>

<SCRIPT Language='javascript' >

function ProcessPart(part)
{
        if (((part.Sender == 1)||(part.Sender == 2)) && (part.Parttype == PARTTYPE_VALUE2))
        {
                if (part.Channel == 1) {RadioButtonSetvalue('LED' + part.Sender, part.Numvalue);};
        }

        if ((part.Sender == 2) && (part.Parttype == PARTTYPE_VALUE2))
        {
                if (part.Channel == 0) {TextboxSetvalue('ADC', part.Numvalue);};
        }
}


function SetSensorScale(quantity)
{
        var parts = InitParts();
        parts = SetScale(parts, 10, 2, 0, 0, quantity, 0, 0);
        SendParts(parts, true);

        if (quantity==RESISTANCE) {TextSetvalue('unit','Ohm');};
        if (quantity==RAWVALUE) {TextSetvalue('unit','ADC-Value');};
}

</SCRIPT>


<FORM Name='Bus'>

<STYLE type='text/css'>#head {font-size:20}</STYLE>

<DIV ID='head' >ElektorBusBrowser </DIV> <br/>

Scheduler
<BUTTON Type='button' onclick='javascript:SetScheduler(SCHEDULER_ON,2,10,0,0,0,0,0,0)' >on</BUTTON>
<BUTTON Type='button' onclick='javascript:SetScheduler(SCHEDULER_OFF,2,10,0,0,0,0,0,0)' >off</BUTTON>
<br/><br/><br/>

LED Node 1
<INPUT Type='radio' ID='LED1' Name='LED1' Value='LED1' />

LED Node 2
<INPUT Type='radio' ID='LED2' Name='LED2' Value='LED2' /> <br/><br/>

<INPUT Type='text' ID='ADC' Value='' /> <SPAN ID='unit' >ADC-Value</SPAN> <br/>

<BUTTON Type='button' onclick='javascript:SetSensorScale(RESISTANCE)'>Ohm</BUTTON>
<BUTTON Type='button' onclick='javascript:SetSensorScale(RAWVALUE)'>Adc raw</BUTTON> <br/><br/>

<BUTTON Type='button' onclick='javascript:GotoUrl("Limit")'>Set-Limit-Page</BUTTON> <br/><br/>

</FORM>
```

Funciones importantes de la librería Javascript JSBus

```
function InitParts()
```
Devuelve una array de partes vacío (llamada: var parts = Initparts();).

```
function SetLimit(parts, sender, receiver, channel, mode, limit, numvalue)
function SetScale(parts, sender, receiver, channel, mode, quantity, unit, scale)
function SetValue(parts, sender, receiver, channel, mode, setvalue)
```
Estas funciones dependen del array parts para cada parte nueva, con los valores de medida/unidad/escala, o el valor fijado de un sensor/actuador, devuelven el array ampliado.

```
function SendParts(parts, overrideQueue)
```
Codifica y envía todas las partes en uno o varios mensajes, ver texto.

```
function PartText(part)
```
Devuelve la representación como texto de una parte, por ejemplo para propósitos de debugging.

```
function RadioButtonSetvalue(id, setvalue)
```
Activa o resetea un radiobotón (setvalue = 1 o 0).

```
function TextboxSetvalue(id, setvalue)
function TextSetvalue(id, setvalue)
```
Describe una Textbox u otro elemento con texto.

```
function GotoUrl(url)
```
Hace que el Host, cargue una nueva página en HTML (url = nombre del archivo sin ".htm").

```
function SetScheduler(status, schedulednode1, … , scheduledNode8)
```
Activa o desactiva el planificador del Host (status = SCHEDULER_ON / SCHEDULER_OFF) e informa al planificador sobre la nueva lista de nodos a los que regularmente tiene que preguntar por envíos (un 0 indica el final de la lista).

Los atributos adicionales del tipo *Name* en un elemento radiobotón (LED1 y LED2) se encargan de que los botones puedan controlarse independientemente. En otro atributo idéntico *Name* nos aseguramos que sólo pueda pulsarse un botón a la vez, como ocurría en las radios antiguas.
El atributo *onclick* del botón llama a la función en Javascript al hacer clic. Los clics en los dos primeros botones activan y desactivan el planificador mediante la función `SetScheduler` (en JSBus.txt). Los botones definidos más abajo llaman a la función específica de la aplicación `SetSensorScale` (ya descrita anteriormente) con sus correspondientes parámetros (`RESISTANCE` o `RAWVALUE`). El último botón se asegura de que el navegador carga correctamente el formulario "Limit.htm". Para ello, se utiliza la función `GotoUrl` en la librería JSBus. Como parámetros se le pasa el nombre del archivo (sin ".htm"). Nótese que en este caso hemos de colocar entre dobles comillas el nombre, ya que las comillas simples ya se utilizan para el valor del atributo *onclick*. Los archivos han de colocarse en el directorio "UIBus" del escritorio. Las siguientes versiones del ElektorBus-Browser permitirán más opciones incluso.

De un vistazo

Aquellos que ya tengan conocimientos sobre HTML/Javascript, podrán junto con esta información y los cuadros consultar las funciones más importantes de la librería JSBus, y podrán empezar rápidamente con sus propias aplicaciones. Los principiantes han de revisar primero el cuadro "Conceptos básicos de HTML/Javascript" (descarga en [2]) y posteriormente practicar viendo los efectos que produce modificar el código HTML. Para ello basta con un sencillo editor de texto, y con un doble clic en el archivo .htm se abrirá en nuestro navegador por defecto. Así podremos ver el aspecto que tendrá el interfaz de usuario posteriormente. Ya que este sistema lo utilizaremos para el resto de entregas de la serie del bus, aparecerán rápidamente otros códigos Javascript para utilizar en nuestras propias aplicaciones.
En cualquier caso, cacharrear con Javascript/HTML siempre es muy útil, ante todo si consideramos que lo siguiente en la serie es implementar nuestro control en un smartphone con Android (Ilustración de apertura). Como puente con el bus dispondremos de una práctica tarjeta basada en el chip Vinculum II de FTDI, que queremos presentaremos en la edición de enero. Esta

tarjeta no sólo sirve para el ElektorBus, sino también para otras muchas aplicaciones. En estas, Javascript/HTML se presenta como una solución independiente de la plataforma para desarrollar interfaces de usuario.

(110517)

¡Forma parte del desarrollo! ¡Cualquier consejo o sugerencia son bienvenidos en el mail de nuestra redacción, redaktion@elektor.de!

Enlaces

[1] www.elektor.es/110258
[2] www.elektor.es/110517
[3] www.elektor.es/110428
[4] www.elektor.es/110382
[5] www.w3schools.com
[6] http://de.selfhtml.org/

Un nuevo bit

Mientras escribía este artículo, seguí los consejos de lectores de Elektor que contribuyeron parcialmente con sus propios desarrollos. Jan Dalheimer de Suecia, a sus 15 años es probablemente el fan del bus más joven. Justo al cerrar la edición Jan acababa de implementar la librería del ElektorBus para controladores AVR. Su primera variante en un principio sólo manejaba el *ElektorMessageProtocol* y el *HybridMode*, y no las últimas partes presentadas del *ApplicationProtocol*. De hecho, Jan fue uno de los que presento su disconformidad con el mecanismo de Acknowledge anterior. Ciertamente es posible marcar los mensajes con la ayuda del *ElektorMessageProtocol*, de modo que podamos recibir confirmaciones del emisor (mediante el Bit0 del byte de modo, véase [1]). Una confirmación de llegada (que envía de nuevo los bytes de datos recibidos al emisor) en este nivel no puede diferenciarse de un mensaje auténtico, y por eso hemos introducido el flag correspondiente primero sólo en el ApplicationProtocol. Sin embargo, se la función del sistema de bus requiere dos pasos en el protocolo, lo cual es un atraso si lo comparamos con implementar limpiamente una librería.

Jan sugirió utilizar el bit1 del byte de modo para distinguir entre el *AcknowledgeMessage* y el mensaje original. Una buena idea que nos apuntamos como posibilidad para marcar los *AcknowledgeMessage* auf en el nivel de los mensajes. Ya que bit1 del byte de modo está cubierto por otra función, hemos de dar al resto de funciones un nuevo orden:

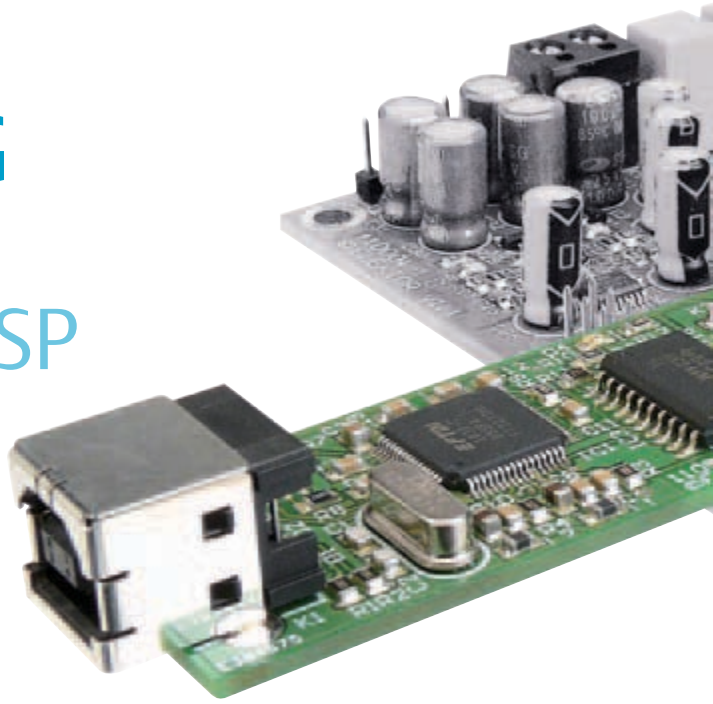
Bit	1	0
7	Sin bytes de ID, datos a partir del byte 2	Bytes de ID a partir del byte 2
6	Los bytes 2 y 3 son de ID	Los bytes 2 a 5 son de ID
5	Sin CRC	Los bytes E y F forman un CRC de 16 bits
4	El último byte de ID es el número de fragmento	Todos los bytes de ID son para el direccionamiento
3	El siguiente fragmento va directamente	No hay siguiente fragmento que le siga
2	Los seis bits de dirección más altos para el segmento del bus	Sin dirección de segmentos
1	AcknowledgeMessage	Mensaje original
0	Esperando al AcknowledgeMessage	No se espera

Bit1 y Bit0 se han marcado en rojo en la figura 5. En la librería de Javascript ya se ha tenido en cuenta la sugerencia previa: una confirmación se hace automáticamente con el byte de modo=2 yse recibe el mensaje original con el byte de modo=1.

Hemos de distinguir el mecanismo de confirmación a nivel del *ApplicationProtocol*; un ejemplo es la alarma de nuestro foto-sensor (véase [3]). Ya que a estos nodos se les pregunta con cierta frecuencia, el byte de modo para los mensajes enviados por él suele ser normalmente 0 (no es necesaria confirmación, ya que no pueden darse colisiones). En este caso, el desarrollador ha de implementar la confirmación por su cuenta. Para distinguirla del mensaje originales se ha utilizado un flag de Ack dentro de los datos útiles [4]. Una Part importante dentro de un Message (por ejemplo, una alarma), codificará fácilmente mediante un Ackflag=true un nuevo mensaje, y lo enviará de vuelta. Al respecto, la librería en Javascript incluye la función `AddAckPart(...)`, podemos encontrar un ejemplo de aplicación en el archivo “Limit.htm”.

Interfaz OnCE/JTAG

Para la programación y depuración de los DSP de Freescale



Ton Giesberts (Laboratorio de Elektor)

El laboratorio de Elektor ha diseñado un pequeño adaptador para la programación y depuración de la tarjeta DSP de nuestro curso DSP, con el que es posible conectar dicha placa con el PC a través de una moderna conexión USB rápida. Este adaptador también se puede utilizar en combinación con otros DSP de Freescale de la serie DSP56K.

Hemos diseñado un circuito para el curso entorno al DSP56374, el DSP de la serie Symphony de Freescale. La programación y depuración se ha hecho a través de un interfaz JTAG, para el que Freescale utiliza un conector propio de 14 vías, llamado OnCE (abreviatura de On-Chip Emulation). Para que la tarjeta DSP fuera lo más compacta posible, no se integró un interfaz directo para la conexión con el PC. El propio Freescale ofrece algunos adaptadores de programación, pero son bastante caros. Quien haya trabajado ya con un kit de evaluación de Freescale, dispondrá, probablemente, de un adaptador de programación apropiado. Además, en Internet puedes encontrar varias soluciones diferentes (de construcción propia) para la programación de estos DSP de Freescale. Sin embargo, la mayoría funcionan a través del puerto paralelo, y hay pocos ordenadores (nuevos) equipados con dicho puerto. Como complemento a la tarjeta DSP publicamos aquí un interfaz OnCE/JTAG con conexión USB, que se encarga de establecer una moderna interconexión rápida entre la tarjeta DSP y el PC.

El circuito

Para el interfaz USB optamos por el integrado FT2232H, un “Hi-Speed Dual USB UART/FIFO” de FTDI (IC1, ver **figura 1**). Este integrado pertenece a un tipo de la generación más reciente de FTDI (ver hoja de características en [1]). A través de este inte-

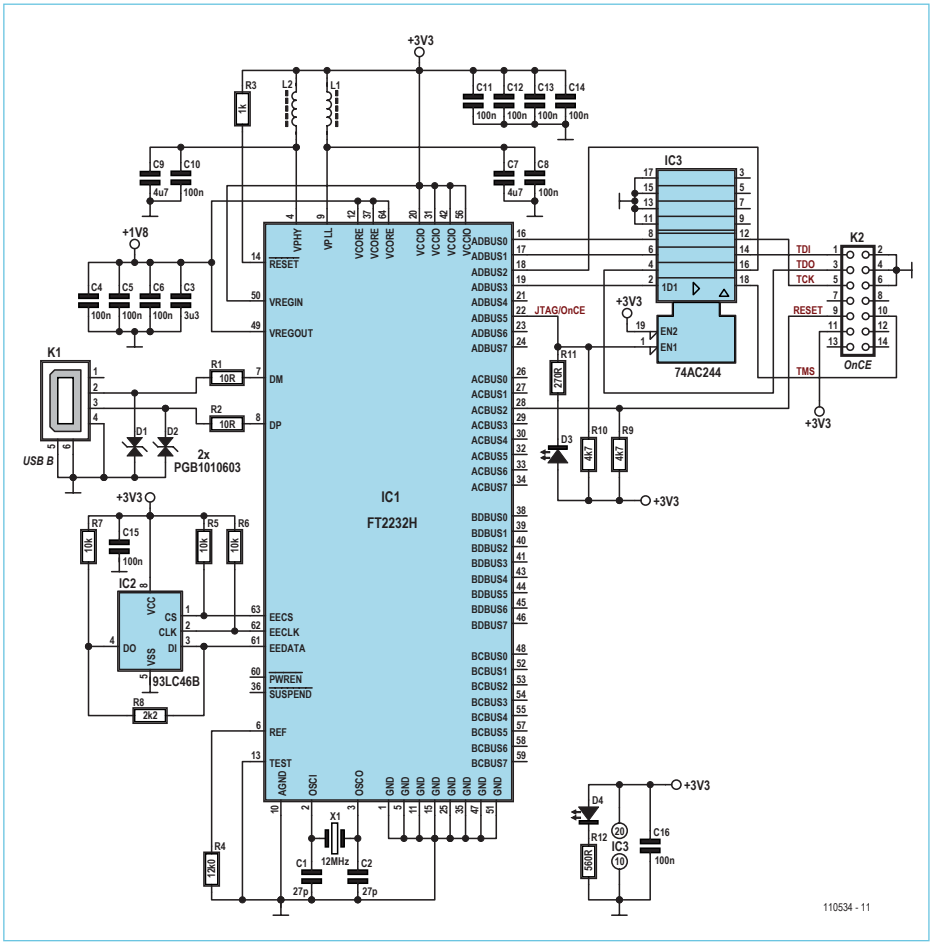


Figura 1. El esquema del adaptador de programación, con un FT2232H de FTDI como integrado principal.

grado, el interfaz puede comunicarse con el PC a la velocidad de un USB2.0 de alta velocidad. La tarjeta DSP suministra la alimentación del circuito, de modo que el interfaz es "Self Powered". Esto hace que el circuito sea más sencillo, porque la alimentación del interfaz de E/S asciende a 3,3 V y ésta ya se encuentra disponible en la tarjeta DSP. El núcleo del FT2232H trabaja a una tensión más baja, es decir 1,8 V. Para eso el integrado tiene incorporado un regulador de tensión propio, el cual empleamos gustosamente. Para la conexión del bus USB no hace falta más que dos resistencias y un conector USB B (K1). Como protección de descargas electroestáticas se han colocado dos supresores ESD (D1 y D2) justo al lado del conector USB. El tiempo de reacción de estos diodos es menos de 1 ns. Debido a su capacidad extremadamente baja (sólo 0,055pF), no tienen ninguna influencia sobre las señales USB. El reset activo a nivel bajo del integrado no se utiliza, y está conectado con la tensión de alimentación a través de R3. La EEPROM externa (IC2) está conectada con el FT2232H según una aplicación estándar (R5...R8). La EEPROM tiene que ser de una versión con un tamaño de palabra de 16 bits, ha de funcionar con una tensión de alimentación de 3,3 V, pero no hace falta que tenga la posibilidad de configurar la memoria (una conexión llamada frecuentemente ORG). El 93LC46B de Microchip es un candidato perfecto para esto. La tensión de alimentación del FT2232H está muy bien desacoplada, a juzgar por los 12 condensadores y dos bobinas (C3...C14/L1/L2).

Hemos optado por dotar de buffers a las señales de salida, excepto a la señal de reset. Las salidas de los buffer están desconectadas y forman una alta impedancia cuando el adaptador aún no se haya activado en el entorno de desarrollo utilizado. El rápido 74AC244 (IC3, un buffer/driver óctuple con salidas triestado) es apto para tensiones de alimentación desde 1,5 hasta 5,5 V y, por lo tanto, funciona perfectamente a 3,3 V. Se ha colocado como protección una resistencia pullup (R9) en la salida reset para que el interfaz funcione sin problemas con otros proyectos. La tarjeta DSP ya lleva una resistencia pullup. Si trabajas únicamente con esta, puedes omitir R9, si quieres.

Puedes ver la placa diseñada para este adaptador en la **figura 2**. Se pudo mantener el tamaño reducido por el uso de SMD. El conector 2x7 está montado en la parte posterior de la placa, de modo que el adaptador se pueda colocar fácilmente sobre el conector K8 de la tarjeta DSP. Aparte de la placa suelta, Elektor provee también de una versión completamente montada y probada de este adaptador [2].

Software para Symphony Studio

Para el diseño del circuito partimos de la situación de que el adaptador se utiliza en combinación con el entorno de desarrollo de Freescale, Symphony Studio. Con la ayuda de una plantilla de Freescale puedes programar el FT2232H de tal forma que sea reconocido como un Symphony SoundBite. Esto ya se ha hecho con la placa completamente construida que ofrece Elektor y puedes pincharla directamente en la tarjeta DSP. Quien quiera programar el FT2232H por sí mismo, puede utilizar para este fin la utilidad de FTDI, MProg. La plantilla de Freescale está hecha para este programa. (FTDI dispone también del sucesor, FT_Prog, pero este no es apto para la plan-

Lista de materiales

Resistencias (SMD 0805, 100 mW):
R1,R2 = 10 Ω 5%
R3 = 1 kΩ 5%
R4 = 12k0 1%
R5,R6,R7 = 10 kΩ 5%
R8 = 2k2 5%
R9,R10 = 4k7 5%
R11 = 270 Ω 5%
R12 = 560 Ω 5%

Condensadores (SMD 0805):
C1,C2 = 27 pF/50 V 5%, NP0
C3 = 3µ3/10 V 10%, X5R
C4...C6,C8, C10...C16 = 100 nF/50 V 10%, X7R
C7,C9 = 4µ7/6,3 V 10%, X5R

Bobinas (SMD 0805):
L1,L2 = 600 Ω @ 100 MHz, 200 mA/0,35 Ω (por ejemplo Murata BLM21BD601SN1D)

Semiconductores:
D1,D2 = PGB1010603, Vclamping = 150 V (Littelfuse, SMD 0603)
D3 = LED verde (Kingbright KPHCM-2012CGCK, SMD 0805)
D4 = LED rojo (Kingbright KPHCM-2012SURCK, SMD 0805)
IC1 = FT2232HL-R (FTDI, SMD de 64 terminales LQFP)
IC2 = 93LC46B/SN (Microchip, SMD SO-8)
IC3 = CD74AC244M (Texas Instruments, SMD SO-20)

Varios:
K1 = conector USB-B acodado para montaje en placa impresa
K2 = conector hembra de 2x7-terminales, paso 2,54 mm
X1 = cristal de cuarzo de 12 MHz, Cload 18 pF ±30 ppm, HC-49S
Placa 110534-1
Placa completamente montada y probada: 110534-91

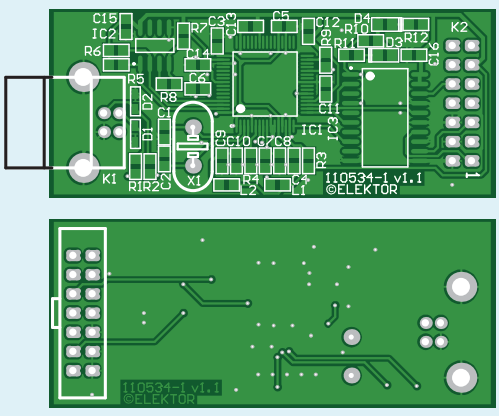


Figura 2. La placa del circuito está diseñada de tal forma que pueda estar conectada directamente sobre el conector K8 de la tarjeta DSP.

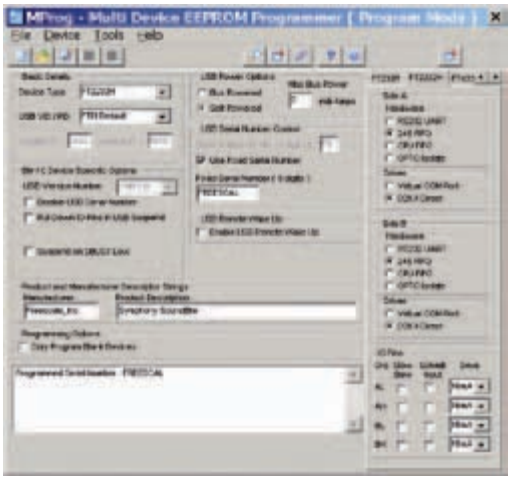


Figura 3. Una captura de pantalla del programa con el que se puede programar el FT2232H.

tilla de Freescale.) En el sitio Web de FTDI todavía puedes descargar la versión más reciente: la 3.5 de MProg [3]. Un problema con este programa es que por desgracia, la versión Single Channel, el FT232H, no se puede programar. Para que la programación del interfaz sea más fácil, para que no tengamos que configurar primero una plantilla propia, hemos optado por la versión (por desgracia más cara) dual-channel. Ya hemos adaptado la plantilla SoundBite original (ver **figura 3**) en varios puntos y puede ser programada directamente. Ejecutando primero un *Scan* (por debajo de *Device*), puedes ver en la ventana de mensajes situada en la parte inferior izquierda si el programa reconoce el interfaz, si efectivamente no está programado o si ha encontrado más interfaces por casualidad. En caso de encontrar el interfaz, puedes programarlo si efectivamente está vacío, ya que en caso contrario habrá que ejecutar primero un *Erase*. Para construir el interfaz y/o programar el FT2232H por uno mismo, hay que hacerlo de la siguiente forma:
Elige en el menú *File* 'Open' y busca la plantilla 110534-1.ept, puedes descargarla de nuestro sitio Web (110534-11.zip, ver [2]). Si quieres modificar la plantilla, elige *Edit* por debajo de *File* después de abrirla. Los siguientes cambios ya se han hecho en la plantilla original de Freescale. Hemos seleccionado el FT2232H en vez del FT2232D como Device Type. En las opciones de USB Power hemos seleccionado Self Powered. Después de seleccionar el FT2232H, aparece a la derecha una pestaña con las configuraciones de los terminales E/S. La elección del Hardware (Side A), es '245 FIFO' en vez del estándar 'RS232 UART'. Los controladores de dispositivos ya están configurados correctamente, D2XX Direct. Asegúrate de que estén instalados [4] estos controladores. En la hoja de características del FT2232H puedes encontrar 2 ejemplos de aplicaciones para el Self Powered. En ambas se utiliza un divisor de tensión para detectar la tensión de 5 V del bus. En el texto se menciona entonces que hay que elegir la opción 'suspend on DBUS7 low' en MProg. Esto sólo funciona si el correspondiente divisor de tensión está presente. Si accidentalmente se hace en nuestro circuito, el PC no reconocerá el circuito. Para poder salir de esta situación, parece que una solución es la interconexión del terminal 46 (BDBUS7) con la tensión de 3,3 V y no la del terminal 59 tal y como indica la hoja de características. En caso de que esto ocurriese, utiliza un hilo de cobre lacado de 0,1 mm, por ejemplo. Ten cuidado de que no haya cortocircuitos, ya que los terminales se encuentran en una máscara de 0,5 mm. Todos los terminales de E/S están configurados con valor máximo de 16 mA. Con esto se controla mejor el LED verde (D3). El consumo del LED asciende a

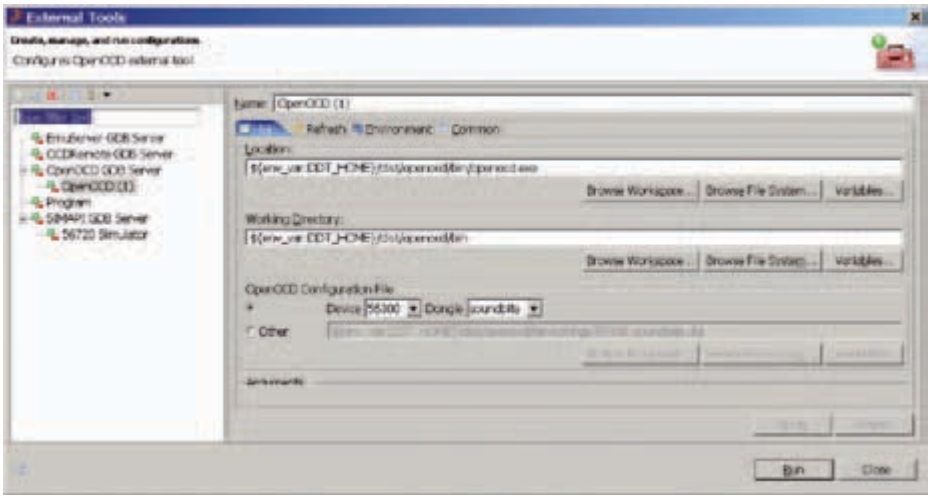


Figura 4. Aquí puedes ver cómo se selecciona el adaptador en el Symphony Studio.

Consumo

Hemos realizado varias mediciones del prototipo en relación al consumo. Si se conecta la tarjeta DSP sin el interfaz a la tensión de 5 V (las 5 V digitales y las 5 V analógicas están interconectadas en la tarjeta DSP a través de una bobina de desacoplo), entonces el consumo asciende a unos 84 mA. Con el adaptador añadido, esto asciende a unos 87 mA después de la conexión. El FT2232H se encuentra aún en modo suspendido y sólo consume varios cientos de µA. En el momento de conectar el interfaz con el PC, el consumo total depende de la velocidad, 135 mA (velocidad completa) ó 155 mA (velocidad alta). Después de configurar el interfaz en el Symphony Studio, el consumo aumenta a unos 160 mA. Estos 5 mA adicionales surgen gracias al LED verde que indica que la salida OnCE está activa. Si se ejecuta, por ejemplo, uno de los programas de prueba, como es el tst_src2.asm, entonces el consumo total (con SRC activado y señales de audio digitales ópticas) aumenta hasta unos 272 mA.

unos 4,7 mA. El consumo del LED rojo de potencia (D4) es menor, unos 2,7 mA, para que la claridad de ambos LED sea más o menos igual para el ojo. Para programar la plantilla modificada habrá que guardarla primero. Siempre es mejor no sobrescribir la plantilla original, elige, por tanto, un nombre nuevo (selecciona *Save As* ... por debajo de *File*). Después de abrir una plantilla puedes programarla directamente.

Ahora hay que seleccionar 'soundbite' como nuestro interfaz en la configuración de Symphony Studio. Por ejemplo en el C/C++ Perspective: Selecciona *Run*, *External Tools*, otra vez *External Tools*, *OpenOCD GDB Server* (con la primera vez hacer doble clic). En la pestaña *Main* hay que seleccionar Device '56300' y el Dongle 'soundbite' en el fichero de configuración de OpenOCD (ver **figura 4**).

Para que la utilización del circuito sea lo más fácil posible, hemos diseñado la placa en forma de módulo, que puede ser conectada directamente sobre la tarjeta DSP. El conector de la tarjeta DSP (K2) está montado en la parte posterior como socket (conector hembra). El conector USB es una versión B estándar para montaje en placa. Preferiblemente debes utilizar un cable USB2 para la conexión, esto está específicamente impreso en el cable.

(110534)

Enlaces Web

[1] www.ftdichip.com/Products/ICs/FT2232H.htm

[2] www.elektor.es/110534

[3] www.ftdichip.com/Support/Utilities/MProg3.5.zip

[4] http://www.ftdichip.com/Drivers/CDM/CDM20814_Setup.exe

Visualizador vocal

¿Y si hiciésemos hablar a nuestro próximo montaje?

Christian Picard (Francia)

A veces es imposible seguir un visualizador con los ojos para vigilar un valor de medida, cuando nuestra atención está puesta en otro lugar. Tan sólo un ejemplo: seguimos la evolución de nuestro equipo de modelismo preferido, por tierra o por aire, estando equipado este último con un dispositivo de telemetría. Sería interesante poder seguir su velocidad, su altitud o el valor de la tensión de su batería de propulsión, y ello tan sólo escuchando. ¿Qué os parece?

Descripción del módulo

La voz del conjunto es un módulo 4D Systems, el SOMO-14D [1], capaz de leer ficheros contenidos en una tarjeta de memoria micro-SD de 2 GB como máximo. La salida de audio del modelo se hace sobre un altavoz o unos cascos.

El cerebro de este montaje es un microcontrolador Atmega8AU, controlado por un cristal de cuarzo de 8 MHz, cuya tarea es la de controlar el módulo de sonido que ejecutará la lectura vocal del número o del mensaje analizado. El módulo de sonido está controlado por un enlace serie de dos hilos (*data* y *clock*), gestionado por el micro, que recibe sus órdenes como esclavo del bus I²C.

Este montaje será conectado sobre el bus I²C de una aplicación y permitirá, por ejemplo, leer en voz alta una cadena de caracteres ASCII que representan un número entero o en coma flotante, positivo o negativo, resultante de un cálculo almacenado en una variable de tipo «string».

El módulo SOMO puede ser utilizado sólo en una aplicación que utilice directamente el bus de dos hilos, en el caso, por ejemplo, de la lectura de ficheros de sonido, pero esto tendrá como consecuencia una ocupación de tiempo en el programa principal y de espacio de la tan preciada memoria, si el micro utilizado está a pleno rendimiento. La ventaja suplementaria del módulo descrito es su portabilidad a otras aplicaciones.

El micro-programa

El programa principal del micro se reduce a un bucle de espera que analizará la cadena recibida, tan pronto como la trama de los datos I²C sea descifrada en su totalidad. En este bucle, los caracteres recibidos activan una interrupción por carácter. El programa saldrá del bucle al final de la cadena, indicado por el carácter «nulo» $00, seguido por dos caracteres hexadecimales que representarán el número (el nombre) del fichero almacenado en la tarjeta micro-SD que contiene la unidad del valor de medida que acaba de ser enunciado (u otro mensaje).

Veo en vuestra mirada una luz de desamparo que voy a intentar apaciguar con un ejemplo concreto. Supongamos que el número que hay que leer es -235,12 V. La cadena de datos ASCII que hay que transmitir se elaborará del modo siguiente (hexadecimal): $2d $32 $33 $35 $2c $31 $32 $00 $00 $69. Debemos haber reconocido el signo menos en el carácter $2d y la coma en $2c. Para el resto de caracteres creo que me habéis seguido salvo, quizás, para los dos últimos. Recordemos que aún queda una unidad de medida por leer: “voltios”. En efecto, $00 $69, 105 en decimal, es el número del fichero “volt” en la tarjeta micro-SD. Al resaltar que esto es en lo referente a mis ficheros, es posible que en vuestro sistema tengáis otro número ya que será tarea vuestra el registrar vuestros propios ficheros de audio.

Volvamos al análisis. La primera de las verificaciones es la de saber si la cadena recibida corresponde a un número o a un direccionamiento directo del módulo SOMO-14D. Si el primer carácter recibido es el carácter nulo ($00), el programa se inclinará hacia el tratamiento de los dos octetos hexadecimales siguientes como un número de fichero y, a continuación se leerá dicho fichero. Si el primer carácter no es el carácter nulo, la cadena es considerada como una representación en ASCII de un número. En ese caso, el programa busca si el número es un entero, un decimal, positivo o negativo, cuantas cifras hay antes de la coma, cuantas después (1, 2 ó 3 cifras como máximo), etc. El encaminamiento hacia los diferentes sub-programas dependerá de estos últimos elementos. Entre los octetos recibidos que no representan números y que no pertenecen a números de ficheros, pueden ser comandos directos con destino al módulo de sonido, por ejemplo, el ajuste del nivel de sonido ($FF $F0 a $FF $F7).

Será necesario hacer una adaptación menor para la utilización del inglés (u otros idiomas), para los números únicamente. Veamos, por ejemplo, el caso particular del francés para las centenas y los

millares. En francés se dice “cent” mientras que en inglés se dirá “one hundred” y el punto sustituirá a la coma. Es sutil, pero justifica una ligera disposición de la estructura del programa. En cuanto a los otros idiomas, reconozco humildemente mi falta de conocimiento.

A señalar...

- Los ceros no significativos al inicio de la cadena serán suprimidos.
- Si un cero (ASCII $30) inicia la cadena, el número será interpretado como que es inferior a uno y se leerá un máximo de tres cifras después de la coma.
- Cuando un número es nulo, la cadena será igual a $30 $00.
- Para un número sin unidad, le carácter nulo de fin de cadena estará seguido de $C0, de esta forma no se leerá ninguna unidad y la presentación sonora será detenida.

Algunos ejemplos

Como el autor es francés, los ejemplos están en su idioma materno.
12 se escucha como «douze».
527 se escucha como «cinq cent vingt sept».
13 689 222 se escucha como «treize méga six cent quatre vingt neuf mille deux cent vingt deux».
0,155 se escucha como «zéro virgule cent cinquante cinq»
-0,155 se escucha como «moins zéro virgule cent cinquante cinq»

El circuito

El esquema eléctrico del montaje se muestra en la **Figura 1**. De tamaño relativamente pequeño (dimensiones: 25 x 40 mm), el módulo encontrará fácilmente su lugar en las aplicaciones. El montaje necesita un poco de destreza, pero tampoco demasiada. La soldadura manual del ATmega8AU (modelo SMD con encapsulado TQFP32) viene facilitada por el uso de pasta para soldar, la misma que se utiliza para la soldadura por refusión. Aquí os dejo el método utilizado por mí: el circuito se coloca con precisión y se sujeta con una pinza con muelle. Después, se aplica la pasta moderadamente sobre los terminales del circuito. Un soldador de punta fina asegurará las soldaduras sin riesgo de cortocircuito entre los terminales.
Un emplazamiento para el soporte HE6 permite la programación del micro controlador.
También son necesarios algunas resistencias y condensadores con encapsulado SMD. Los dos diodos 1N4001 (D2 y D3) son los recomendados por el montador para permitir alimentar al módulo de sonido con 5 V (nominal 3,3 V).
El módulo puede ser implantado directamente por soldadura sobre el circuito o por medio de tiras de terminales con contactos tipo “tulipán”.
La salida J9 está destinada a recibir un altavoz o unos cascos de 8, 16 ó 32 Ω. El módulo de sonido puede proporcionar hasta 250 mW.
D4, un LED de 3 mm, muestra cuando el módulo SOMO está ocupado (*busy*).
La comunicación con el bus I²C puede hacerse por medio de hilos o por conectores de terminales SIL. La asignación de las conexiones de J11 es la siguiente:

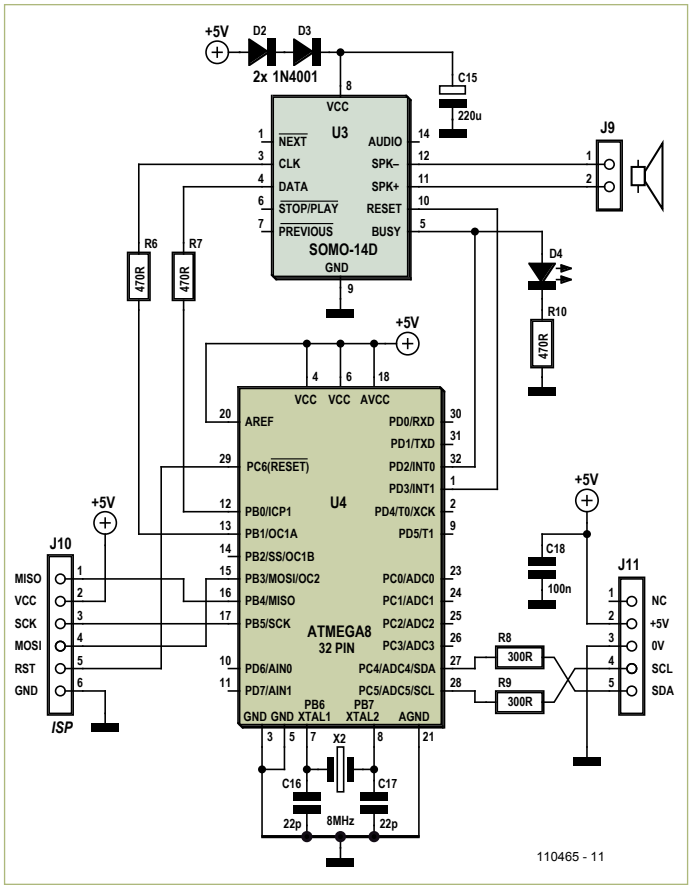


Figura 1. Esquema eléctrico del visualizador vocal.

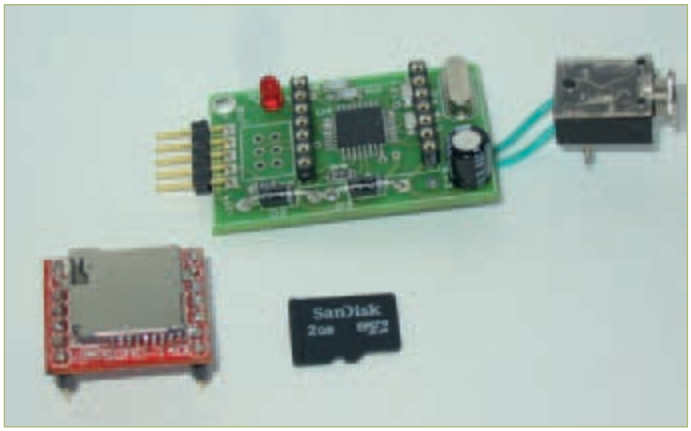
Terminal	Función
1	no conectado
2	+5 V
3	0 V
4	SCL
5	SDA

Todo se almacena en la tarjeta micro-SD

La memoria contiene los ficheros necesarios para la elaboración de los números y de los mensajes. La capacidad relativamente grande de la memoria micro-SD, de 2 GB máximo, permite un uso en diversos dominios. Son necesarios cerca de 105 ficheros para la elaboración de los números en francés. Así pues, quedan 407 ficheros disponibles para el usuario, los cuales podrán contener las diferentes unidades, palabras, frases, canciones, mensajes de alarma para una mayor amplitud de aplicaciones.

Comandos I²C

La dirección del módulo es $58. El protocolo de acceso es estándar, una START seguida de la dirección del módulo, después los datos que representan los caracteres de la cadena a tratar que termina con un comando de STOP.
Los comandos directos no han sido todos comprobados por mí. Para el ajuste del nivel de audio se valida el comando, así como para los direccionamientos directos de los ficheros de sonido.



Los componentes principales del visualizador vocal.

Montaje ensamblado.

Comandos del módulo SOMO-14D según nota del fabricante		
Código comando	**Función**	**Descripción**
0000h – 01FFh	Número del fichero de audio	Direcciona uno de entre los ficheros pre-registrados de audio/sonido/voz en la tarjeta de memoria micro-SD (de 0 a 512 ficheros máx.).
FFF0h – FFF7h	Volumen	Código de ajuste del volumen, 8 niveles en total.
FFFEh	Lectura/pausa	Lee o hace una pausa con el fichero de audio de trabajo.
FFFFh	Stop	Detiene la lectura del fichero de audio de trabajo y coloca al módulo en modo idle.

Detalles sobre los ficheros

Los ficheros elementales en francés utilizados por el programa de análisis pueden ser descargados de [2]. Están preparados para ser volcados sobre una tarjeta de memoria micro-SD. Sin embargo, el propio usuario puede crear sus propios mensajes y los ficheros elementales podrán ser sustituidos por los ficheros del usuario.

Los ficheros elementales representan las cifras de «cero» a «cien», «mil» (0101d), «millón» (0144d), «coma» (0129d), «punto» (0130d) y «menos» (0102d). Con relación a las exigencias del programa, las cifras de 0 a 99 ocuparán los ficheros de 0 a 99, que serán convertidos en caracteres hexadecimales por el programa para direccionar el módulo de sonido. Este comportamiento sólo vale para la parte concerniente a la lectura de los números ya que, como se ha explicado más arriba, los comandos directos de los mensajes serán enviados en hexadecimal.

Las voces utilizadas podrán ser sintetizadas o grabadas en vivo. La única imposición es que las grabaciones necesitan una longitud mínima para ser reconocidas, ya que los ficheros demasiado cortos serán ignorados por el módulo SOMO.

En la página web del fabricante podemos descargar el programa gratuito (SOMO Tool) [3] que permite convertir los ficheros de audio, .wav o .mp3, en .ad4, el único formato reconocido por el módulo SOMO.

El fabricante señala que ciertas tarjetas micro–SD no son válidas. Por mi parte, he utilizado tarjetas de SanDisk que, hasta el presente, han sido totalmente satisfactorias, siempre y cuando se respete la longitud suficiente de los ficheros.

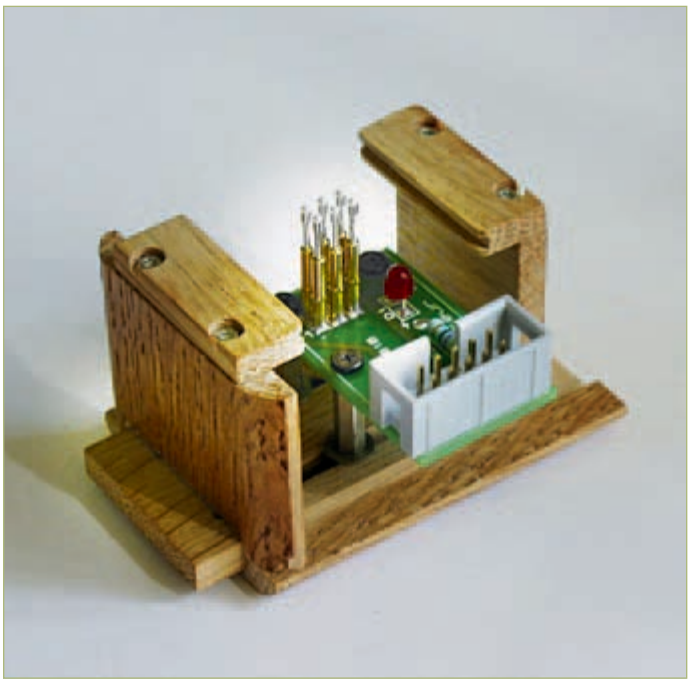
(110465)

Enlaces en internet

[1] Módulo SOMO: www.4dsystems.com.au/prod.php?id=73

[2] Programa y ficheros de sonido en francés: www.elektor.es/110465

[3] Utilidad de conversión de audio SOMO: http://www.4dsystems.com.au/prod.php?id=74



Para los circuitos de pequeño tamaño, la colocación de un conector de programación puede ser un inconveniente. Para evitar este problema, el autor ha creado un dispositivo de programación.

Álbum de Tubos Exóticos

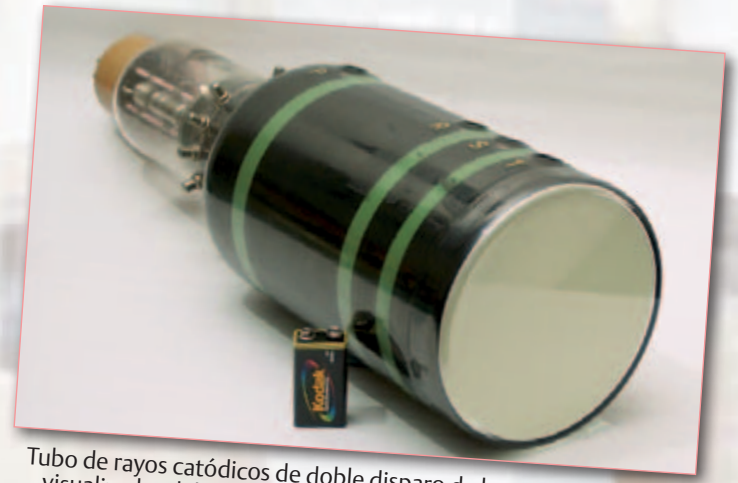
Reginald Neale (USA)

Después de las entregas XXL con los mencionados textos y teoría publicados en los dos meses anteriores, Retrónica se relaja un poco para realizar una gira visual sobre los tubos o, si lo desea, un viaje al fondo del Tube Memory Lane (o Tubo de Línea de memoria). ¿Cuál de los tubos mostrados aquí recuerdan nuestros lectores, vagamente o de manera intensa? ¿Cuáles acumulan polvo en nuestro desván? Y, para los calentadores de tubos: ¿cuáles nos negamos a aceptar?

Uno de nuestros donantes/colaboradores, Reginald Neale, escribe desde Farmington, NY: "Cuando comencé mi carrera de ingeniería, a principios de los años 50, todo estaba hecho con tubos de vacío (y llenos de gas). Los transistores eran aún una curiosidad de laboratorio. Así pues, aquí tenemos algunos de los modelos más exóticos y elegantes."

(110387)

Los tubos de los años 30, como el que se ve a la izquierda, tenían 4, 5, 6 ó 7 terminales base de diferentes tamaños. A continuación vino la base octal codificada, la miniatura de 9 y 7 terminales, el "compactron" de 12 terminales y los modelos sub-miniautura.



Tubo de rayos catódicos de doble disparo de la casa DuMont, del visualizador del radar WWII. La cara del tubo es de unas 5" de diámetro. Un haz escribe las informaciones del texto/escala y el otro escribe la imagen de eco del radar.

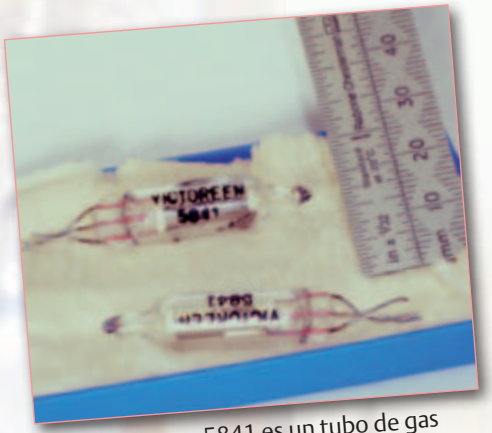
Un tubo de "bellota". Con su construcción plana y sus terminales radiales, fue diseñado para la banda de Ultra Alta Frecuencia. A finales de los años 40, la UHF era la frontera de la alta frecuencia.

Thyratrons es el tubo equivalente del Silicon Controlled Rectifiers (SCRs). La conducción se inicia por una rejilla de control y continúa hasta que la corriente de ánodo-cátodo cae a cero.

Un gran número de grandes nombres en la industria de radio/televisión/tubos ha desaparecido. ¿Quién recuerda a CBS-Hytron, Philco, Stromberg-Carlson, Capehart, National Union, Admiral, Emerson, Tung-Sol, Sylvania, Muntz, Dumont, Wells-Gardner, Crosley, Raytheon, Motorola, Zenith?

Tubo de transmisión/recepción de un sistema de radar WWII. Montado sobre la guía de onda y disparado por un pulso de alto voltaje, cortocircuita un ánodo en la guía de onda, realizando una conmutación efectiva entre los modos de transmisión y recepción, a alta velocidad.

Un tetrodo 4X150 de un tubo transmisor de potencia. El ánodo de metal tiene aletas integradas para el enfriamiento con aire forzado. Fue my famoso: «todo radioaficionado tenía uno».



El Victoreen 5841 es un tubo de gas regulador de alta tensión, sub-miniatura. Regula a una tensión de 900 VDC para una carga comprendida entre 5-100 µA.

Dentro de este tubo hay un calentador y un termopar que genera una débil tensión dependiente del vacío. Una NPT de 1/8" montada sobre el mismo, conecta con el sistema de vacío.

Este tubo de rayos X de los años 50 tiene una longitud aproximada de unos 33 cm. Dispone de un ánodo fijado y un cátodo. Los tubos modernos incrementan la potencia usando un ánodo largo que gira a alta velocidad.

Los fototubos "Miniatura" fueron una de las innovaciones que hicieron posibles las películas con sonido en los años 30. Las pistas de sonido ópticas grababan la información de audio en el borde de la película. Los sistemas de sonido posteriores usaban tiras magnéticas, lo que se tradujo en una mejora de la calidad de sonido.

El tubo de transmisión 832-A de doble tetrodo. Un viejo favorito entre los aficionados a la radio a diferencia del modelo europeo QQE06/40, no estaba neutralizado (ver también el número de Retrónica de diciembre de 2008).

Retrónica (Recuerdos de electrónica) es una columna mensual que cubre equipos electrónicos antiguos, incluyendo diseños legendarios de Elektor. Se agradecen contribuciones, sugerencias y peticiones; por favor, enviad un correo electrónico (email) a redacción@elektor.es