CIRCUIT CELLAR

THE WORLD'S SOURCE FOR EMBEDDED ELECTRONICS ENGINEERING INFORMATION

DECEMBER 2011
ISSUE 257

PROGRAMMABLE LOGIC

MCU-Based Sleep-Stage Analysis

DIY Smart Electronic Load Design

Embedded Linux System Platforms

Interference Immunity for Electronic Designs

Joystick Control with Bluetooth Connectivity

PLUS

Micro Design
Inside the Mind of a Microprocessor Developer

// Analog Bipolar Tech to Digital Design
// The Usefulness of Verilog HDL
// Design Using Programmable Logic
// And More

7 25274 75349 9
$7.50 U.S. ($8.50 Canada)
www.circuitcellar.com

TASK MANAGER

Embedded Community

Up for a quick game? Let's play with the words **circuit** and **cellar**:

> If you enjoyed reading about **The Circuit Cellar** in Steve Ciarcia's **"Circuit Cellar"** column back in the 1980s, then today you likely read **Circuit Cellar, Inc.**'s ***Circuit Cellar*** magazine in your own **circuit cellar**.

Are those typos? Let me explain.

What do you think of when you see the word **circuit** next to the word **cellar**? What's the difference between **The Circuit Cellar** and **a circuit cellar**? What if you capitalize the first letter of each word and put them in quotes (i.e., **"Circuit Cellar"**)? What do you think of when you remove the quotes (i.e., **Circuit Cellar**)? What if you italicize the two words (i.e., ***Circuit Cellar***)?

OK, enough of the editor's game. I have an actual point to make. During the last three decades, the term **circuit cellar** has referenced many different, yet related, things. And, yes, punctuation, grammar, and context all matter.

First, note that **The Circuit Cellar** has always been, and still is, a physical location at Steve Ciarcia's house. I've been there. It's full of electronics, old parts, and cool gadgets.

Next, there's **"Circuit Cellar"** with quotation marks. In the mid-'80s, the term referred to Steve's famous column in *Byte* magazine. Later, after Steve left *Byte*, the term quickly took on a few new meanings all at once. When italicized, the term referred to the internationally respected magazine you're reading today: ***Circuit Cellar***. And an entity had to publish the magazine, right? Hence the company name **Circuit Cellar, Inc.**

And then there's **circuit cellar**. Once Circuit Cellar, Inc. started reaching thousands of readers around the globe with its content, engineers began referring to their own workspaces as circuit cellars. That's right. Years before the term "hackspace" became fashionable, designers were using what they'd read in *Circuit Cellar* to complete innovative projects in their own circuit cellars.

As we wrap up another great year of publishing, I encourage you to reflect on the various ways the term **circuit cellar** has evolved during the past 30 years. It has come to mean so many important things to so many people, from professionals to academics. This positive idea—the notion that our company supports and educates the embedded design community—motivates our staff to strengthen our content and broaden our offerings each year.

On behalf of the staff, I thank all the authors who contributed articles during the past 12 months. I also thank our members who read *Circuit Cellar* each month, chat with our editors, and interact with our authors. I appreciate your interest and support.

FYI: I purposely referred to you as our **members** rather than **subscribers**. Why? You are truly part of a community—The Circuit Cellar Community—and it's a fellowship you help make fresh and exciting every day.

The Circuit Cellar Community? Hmm. I think we now have another term to discuss in 2012. Let me know what you think.

cj@circuitcellar.com

C. Abate

CIRCUIT CELLAR®

THE WORLD'S SOURCE FOR EMBEDDED ELECTRONICS ENGINEERING INFORMATION

FOUNDER/EDITORIAL DIRECTOR
Steve Ciarcia

EDITOR-IN-CHIEF
C. J. Abate

ASSOCIATE EDITOR
Nan Price

CONTRIBUTORS
Jeff Bachiochi
Bob Japenga
Robert Lacoste
George Martin
Ed Nisley
Richard Wotiz

PROJECT EDITORS
Ken Davidson
David Tweed

PUBLISHER
Hugo Van haecke

ASSOCIATE PUBLISHER
Shannon Barraclough

CUSTOMER SERVICE
Debbie Lavoie

CONTROLLER
Jeff Yanco

ADMINISTRATIVE COORDINATOR
Valerie Luster

ART DIRECTOR
KC Prescott

GRAPHIC DESIGNER
Grace Chen

Circuit Cellar's mission is to collect, select, and disseminate need-to-know information around the world in the fields of embedded hardware, embedded software, and computer applications. Circuit Cellar uses an assortment of print and electronic content-delivery platforms to reach a diverse international readership of professionals, academics, and electronics specialists who work with embedded, MCU-related technologies on a regular basis. Our aim is to help each reader become a well-rounded, multidisciplinary practitioner who can confidently bring innovative, cutting-edge engineering ideas to bear on any number of relevant tasks, problems, and technologies.

ADVERTISING
800.454.3741 • 978.281.7708 • www.circuitcellar.com/advertise

ADVERTISING REPRESENTATIVE
Peter Wostrel
Strategic Media Marketing, Inc.
1187 Washington St., Gloucester, MA 01930 USA
800.454.3741 • 978.281.7708
peter@smmarketing.us • www.smmarketing.us
Fax: 978.281.7706

ADVERTISING COORDINATOR
Kim Hopkins
E-mail: khopkins@circuitcellar.com

Cover photography by Chris Rakoczy—Rakoczy Photography
www.rakoczyphoto.com

PRINTED IN THE UNITED STATES

CONTACTS

SUBSCRIPTIONS
Information: www.cc-access.com, E-mail: subscribe@circuitcellar.com
Subscribe: 800.269.6301, www.cc-access.com, Circuit Cellar Subscriptions, P.O. Box 5650, Hanover, NH 03755-5650
Address Changes/Problems: E-mail: subscribe@circuitcellar.com

GENERAL INFORMATION
860.875.2199, Fax: 860.871.0411, E-mail: info@circuitcellar.com
Editorial Office: Editor, Circuit Cellar, 4 Park St., Vernon, CT 06066, E-mail: editor@circuitcellar.com
New Products: New Products, Circuit Cellar, 4 Park St., Vernon, CT 06066, E-mail: newproducts@circuitcellar.com

AUTHORIZED REPRINTS INFORMATION
860.875.2199, E-mail: reprints@circuitcellar.com

AUTHORS
Authors' e-mail addresses (when available) are included at the end of each article.

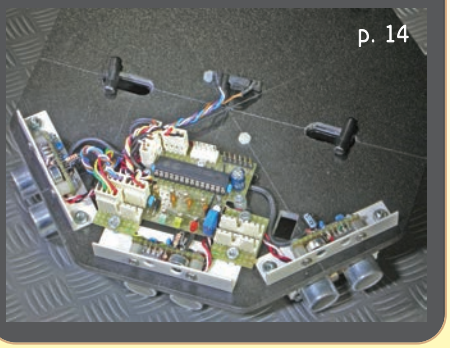
CIRCUIT CELLAR® (ISSN 1528-0608) is published monthly by Circuit Cellar Incorporated, 4 Park Street, Vernon, CT 06066. Periodical rates paid at Vernon, CT and additional offices. **One-year (12 issues) subscription rate USA and possessions $45, Canada/Mexico $60, all other countries $63. Two-year (24 issues) subscription rate USA and possessions $80, Canada/Mexico $110, all other countries $116.** All subscription orders payable in U.S. funds only via Visa, MasterCard, international postal money order, or check drawn on U.S. bank. **Direct subscription orders and subscription-related questions to Circuit Cellar Subscriptions, P.O. Box 5650, Hanover, NH 03755-5650 or call 800.269.6301.**

Postmaster: Send address changes to Circuit Cellar, Circulation Dept., P.O. Box 5650, Hanover, NH 03755-5650.

INSIDE ISSUE 257

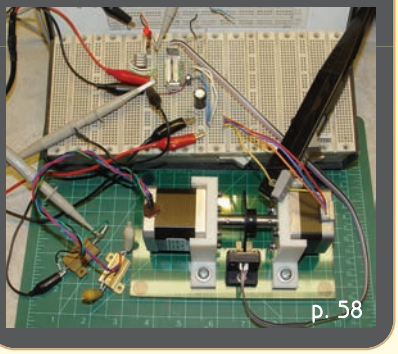
December 2011 • Programmable Logic

Sound Board & Sensors for Programming

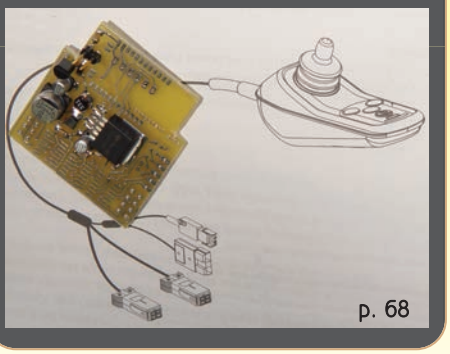


DIY Smart Electronic Load

Dynamometer with Two Steppers



Control System Design



by Guido Ottaviani (Italy)

Sound Tone Detection with a PSoC (Part 2)

The Software & Sensing

The first part of this two-part series describes the hardware part of an audio sensor system. This article covers the design's software capabilities and how they can be used to measure the level of the signal, expand the dynamic of the amplifying chain, and reliably share this information in several different modes with the external world.

In the first part of this article series, I described the "hardware" capabilities of a Cypress Semiconductor PSoC 1 device and how to configure the modules inside to add a new sensor type to my Rino robotic platform (see Photo 1). The previous article shows how to multiplex three microphones on an amplifying chain and how to use a narrow band-pass filter to precisely detect a tone. A peak detector converts the signal to a DC level. Via an analog-to-digital converter (ADC), the software can read this level, act to optimize the programmable gain amplifiers (PGAs) to have the best dynamic, and send the measured level to the sensor board or whatever is waiting for this information with some different communication protocols.

Let's start with the software description.

SOFTWARE

After the analog blocks are connected, they are ready to condition the input signal to have a DC level proportional to the 4-kHz tone intensity. The digital blocks are ready to output the result in different ways. We now need some other glue to attain our goal. Some simple program code connects these two parts of the system returning the data we need.

The designer does most of the initializing job. When you have created the entire circuit, naming, connecting, and configuring all the blocks by selecting the "generate configuration files" menu item, the designer will create all the header, library, and include files needed, as well as the very basic main.c template to start writing your code.

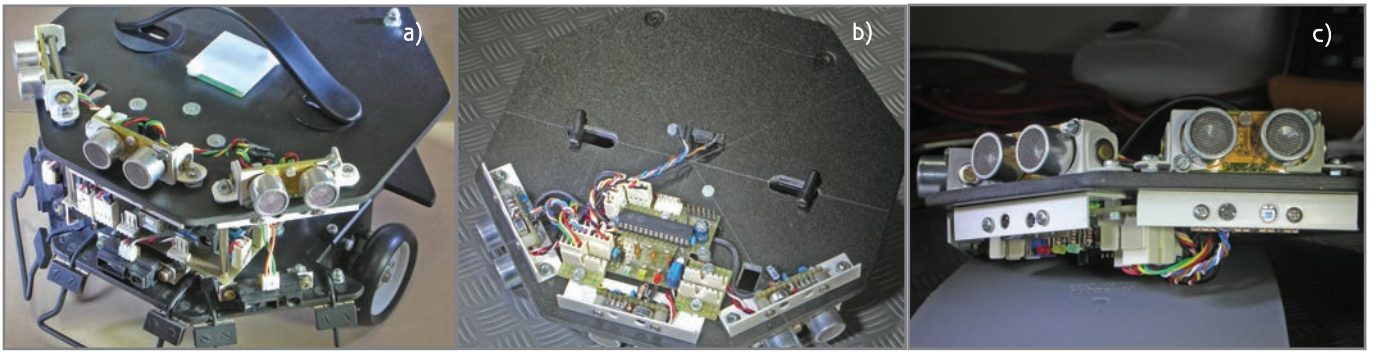


Photo 1a—An overall view of the Rino robotic platform with all its sensors: digital compass, ultrasound sonars, IR distance meters, and light and sound receivers. b—An upside-down view of the PSoC sound board installed among some other boards under the robot top cover. Maybe now it's clear why the platform is so strangely shaped. The board is connected to an I²C hub where all other sensors are connected. The white supports keep the microphones used by the sound board in place. c—Another view of the sensors installed on the robot top cover. Ultrasound sonars and one kind of light sensor are at the top, microphones and another kind of light sensor are under the cover.

If an interrupt is defined, it creates two library files for the module involved. For example, for the Timer module, it creates HB_Tmr.asm and HB_TmrINT.asm on which you must insert your code to start the interrupt service routine (ISR).

With the timer that generates an interrupt every 10 ms we have the "heartbeat" of the program. The ISR updates all the counters and flags used by other procedures to execute time-related actions.

READING THE ADC

This is another improvement. We don't have just a digital output that is only On/Off, but an analog output proportional to the input level that enables some more options with the software.

For each 10-ms timer cycle, the program performs one measure for the trimmer position. The three measures for the analog levels are performed alternatively, switching the three inputs every 30 ms. This is needed to enable the capacitor voltage to stabilize before readings. Every 10 cycles (i.e., every 100 ms) the program sends the values to the serial port.

AUTOMATIC GAIN CONTROL

By knowing the level of the signal at the output of the amplification chain and the gain of the two amplifiers, we know the actual level of the signal at the input. Therefore, we can adjust the gain of the amplifiers to avoid saturation of these stages (i.e., signal with a peak level too close to VDD). If the level goes above a given high threshold or below a low threshold, the gain will be adjusted to have maximum dynamics without saturation (see Figure 1).

SENDING TO THE UART

Using the library functions, it is easy to send data over the serial port. The high-level functions available are well described in the application programming interface (API) section of the datasheet. As explained in the datasheet, the API routines are part of the user module. Thus, you can deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the "include" files.

I've used the library functions a lot during the software development for debugging. It's easy to send the value of some variables of something similar to a communication program to monitor how the program acts.

After that, I'm using an I²C interface to communicate with the sensors board. But, even with a simple protocol, the same data can be exchanged through the UART. Refer to dsNav Communication, "Protocol Description," 2009 (www.guiott.com/Rino/CommandDescr/Protocol.htm).[1]

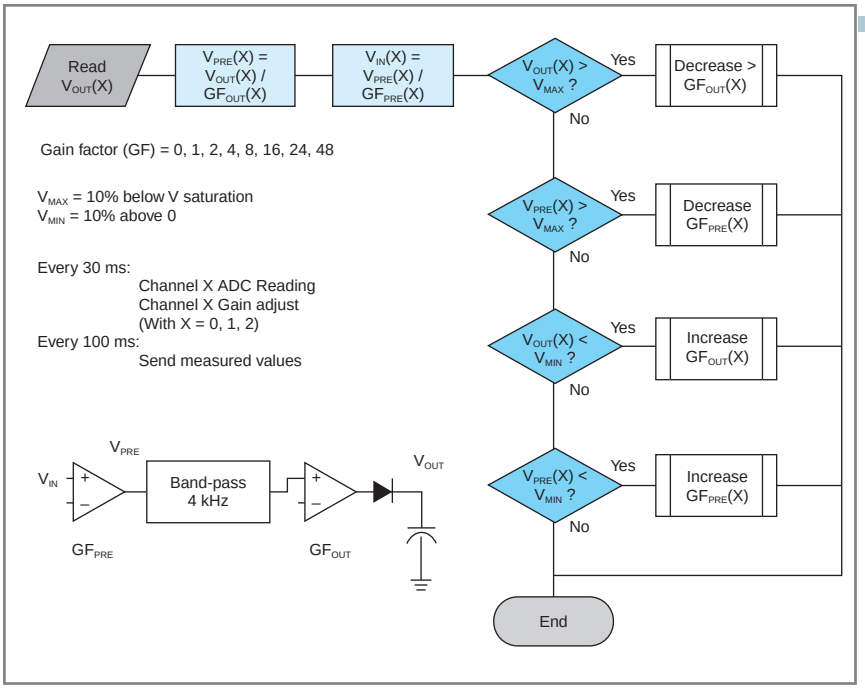


Figure 1—The procedure used for each of the multiplexed channels to adapt the gain of the two amplifier stages to the input sound level (AGC) in order to obtain the best dynamic and avoid saturation.

DRIVING LEDs & DIGITAL OUT

This part of the circuit is just used as backward compatibility with the old op-amp-based board. With the trimmer connected to an ADC port, you can manually set a threshold. When one input level goes over that threshold, the corresponding LED is switched on. The digital output port is set on if one or more LEDs are on.

USING I²C

At the beginning, I placed a standard I²C_HW module. This one is powerful and flexible with many options and slave, master, and multimaster capabilities. In a Cypress Semiconductor application note, there is a project for a bootloader on an I²C bus that becomes transparent to the circuit after program loading.

With all these features, it is not so simple to configure and manage. There are a lot of examples to study; but, in any case, you must still set and control many flags at some specific moments.

After a better analysis of my needs, I decided instead to use Cypress Semiconductor's EzI2C Slave 1.60 module, where "Ez" sounds like "easy." It really is easy, conciliating many people who are overwhelmed by the I²C management. It works only as a slave, with the same protocol of I²C EEPROM. Exactly what I need!

It is enough to place the module (no boxes occupied) and define a data structure that remaps the single-byte registers exposed to the I²C master:[2]

```
struct I2C_Struct { // I2C interface structure
  long I2C_MesValue;
} I2C_Regs
```

Initialize the module:

```
EzI2Cs_1_SetRamBuffer(sizeof(I2C_Regs), 0,
(BYTE *) &I2C_Regs)
```

And start it:

```
EzI2Cs_1_Start(); // Turn on EzI2C
```

In the initialization code, you must configure which registers are used, how many are read/write or read-only, and define a pointer to the data structure. That's all! The registers read and write transparently without any intervention.

The variable can be read by the master pointing at the specific register as a byte in the same way as an I²C EEPROM. Optionally, the ROM area can also be initialized with an equivalent syntax.

With the AGC, I expanded a lot the dynamics of the circuit, therefore "long" variables (4 bytes) must be used. This is exactly what is shown in the previous example. On the master side of the communication, you can use a similar structure to "repackage" the single-read bytes in 4-byte variables, but (there always is a but) you must be careful about the "Endianess" of your microcontroller. I'm using an Arduino board as a sensor board that collects all the interactions with the environment. It is the I²C master and it uses an Atmel microcontroller. This microcontroller uses a little Endian byte order: least-significant byte (LSB) first.

A PSoC uses a big Endian byte order: most-significant byte (MSB) first. Without an appropriate byte-order management you will have a reverted order with very strange values.

A valid alternative to the long variables is the usage of logarithmic values for the sound level instead of linear values. This is mostly used in the audio field and enables you to compress a long variable into a single byte, still maintaining a good resolution on low-level signal measurement. The dynamic range is 255 dB. It is difficult to design a circuit with so huge a dynamic. The range of this PSoC circuit, from the noise to the electret microphone saturation, is about 50 dB.

WATCHDOG

According to the I²C bus specifications, the slave must acknowledge the master exactly within the next clock cycle after the received message packet. To enable the use of I²C communication on a microcontroller that can't guarantee a deterministic response, the slave microcontroller can slow things down. The slave can hold the bus CLK line low until it is ready to send an ACK and reply to the master. This is the so-called "clock-stretching technique," and it hangs up the bus for all the time the microcontroller decides. Of course, the PSoC EzI2C module acts exactly in this way. It's very useful but very dangerous too. If the microcontroller, or its I²C module hangs for any reason, all the I²C devices on the bus will hang.

Because my robot must be reliable enough to perform a mission to Mars beside Spirit rover, I cannot accept this eventuality, even if it has a one in a million occurrence rate.

Enabling a watchdog on the PSoC, I'm sure this cannot happen. Or, if it occurs, the PSoC is reset, releasing the bus and starting from the beginning, missing just some measurement cycles.

The watchdog is enabled in the general preferences using designer IDE. It uses the same timer as the Sleep timer, dividing it by three (i.e., with an 8-Hz sleep timer, the watchdog resets the processor if not cleared for more than 375 ms).

To have complete supervision of the I²C bus, the master (sensors board) pats the dog to keep it awake by writing a specific slave register with a nonzero value every 100 ms. In the main 100-ms cycle of the PSoC program, this register is read. If the value is different from zero, the register and the watchdog timer (WDT) are cleared. If no WDT clear occurs for more than 375 ms—because of errors on the communication bus or because of an incorrect program behavior—the watchdog resets the microcontroller, resolving any problem.

INTERRUPT MANAGEMENT

Let me also criticize something a little bit. The PSoC is a microcontroller; therefore, it is possible to use interrupts in your program. This time too, the designer helps you to create the basic structure to manage the interrupt. But the entire process of assigning an interrupt to a module is not so easy and linear. Once defined, it works fine, but I've seen better ways to do it. In the hopes that this can help someone, I created a brief description of the operations to do in the exact sequence (see in Figure 2).

If you change the name of your block, you must restart everything from the beginning. A new file INT.asm is generated, and your code must be rewritten.

If you don't declare the ISR as an ISR with the `#pragma` directive, the compiler doesn't return any error, but the program will hang. The interrupt priority depends on

To assign an Interrupt Service Routine (ISR) to a block named e.g.: blk1, we must:

- Enable the global interrupt for the project using Designer configuration menu
- Place the blk1 module and enable its own interrupt management on properties windows, specific for each different kind of module
- Compile
- Look for ProjectName/lib/Library Source Files/folder in the workspace explorer window. After compilation, find the file "blk1INT.asm"
- Insert the following text after "Put your code here": "`ljmp _blk1_ISR_C`" exactly with this syntax, with every underscore included
- Come back to C code
- Enable global interrupts with the directive `M8C_EnableGInt`
- Insert the compiler directive: "`#pragma interrupt_handler blk1_ISR_C`" on main.c coded (without first underscore)
- Create your ISR. It must be named `blk1_ISR_C`
- End

Figure 2— A brief description of the operations (in the exact sequence)

block position, according to a list that can be found in the 500-page technical reference manual. It could be easier!

BUILD A BOARD

You can now use this information to build an identical board if you need exactly the same behavior. Or you can use what you've learned to achieve your own goals. I hope this article series was useful to start using PSoC devices to develop similar circuits, or at least to "break the ice" with them.

Author's Note: The complete and up-to-date PSoC Designer project can be found as open source material on my Google Code space.[2]

Guido Ottaviani (guido@guiott.com) has made electronics and ham radio his hobby since the "tube times." He turned the hobby into a job working as an analog and digital developer for several years for an Italian communication company. Many years ago, a big change made him a technical manager at a company that develops and manages graphic, pre-press and press systems, and technologies for a large Italian editorial group that publishes sports newspapers and magazines. Some years ago, he got the scope and the soldering iron out of the drawer to make autonomous robots. Currently, Guido is an active member in various Italian robotics groups who shares his experiences with other self-professed "electronics addicts" and evangelizes amateur robotics.

PROJECT FILES

To download the "ADCINVRcalc.xls" spreadsheet for the calculation of parameters, go to ftp://ftp.circuit cellar.com/pub/Circuit_Cellar/2011/257.

REFERENCES

[1] dsNav Communication, "Protocol Description," 2009, www.guiott.com/Rino/CommandDescr/Protocol.htm.

[2] Google, Explorersound, http://code.google.com/p/explorersound.

RESOURCES

Cypress Semiconductor Corp., "PSoC 1 Overview," 2011, www.cypress.com/?id=1573.

———, Cypress PSoC 1 Application Notes Database, www.cypress.com/?app=search&searchType=keyword&keyword=&rtID=76&id=1573&applicationID=0&l=0.

G. Ottaviani, "Robot Navigation and Control (Part 1): Construct a Navigation Control Subsystem," *Circuit Cellar* 224, 2009.

———, "Robot Navigation and Control (Part 2): Software Development," *Circuit Cellar* 225, 2009.

———, "A Sensor System for Robotics Applications," *Circuit Cellar* 236, 2010.

Rino Robotic Platform, www.guiott.com/Rino/index.html.

SOURCES

Arduino board
Arduino | www.arduino.com

CY8C29x66, CY8C29466-24PX, CY8C27143-24PXI, AN1683, and AN2223 PSoCs, and EzI2C Slave 1.60 component
Cypress Semiconductor Corp. | www.cypress.com