Temi

A meteorological marionette

By **Lars Lotzenburger** (Germany)

Electronics is everywhere in daily life, and even the young cannot be spared the effects of the march of progress. This project illustrates how the latest technology can be used to update the traditional jumping jack toy: instead of pulling a string, 'Temi' is operated by pressing a button, whereupon he will display the current temperature or humidity.

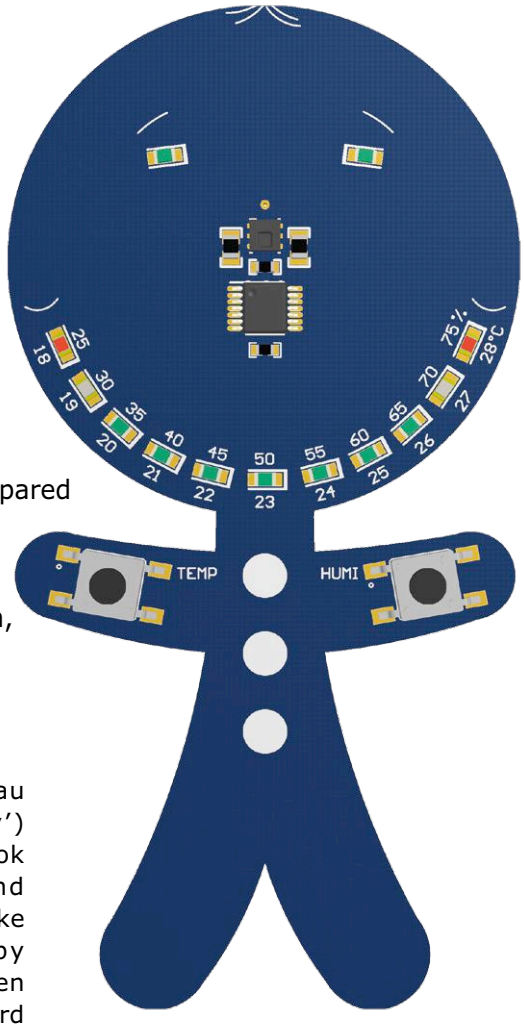


Figure 1. Rendering of Temi's printed circuit board.

It is a pity that an ever diminishing number of people are aware of the capabilities of modern electronic technology, and in particular microcontrollers, and the opportunities it can offer. Perhaps one reason behind this is that ICs and other special-purpose components have become increasingly tiny, and at the same time so complex that it is hard to understand their internal workings. But it is never too early to try to pique a young person's interest in technology, and the aim of Temi is to attract inquisitive but shy fingers.

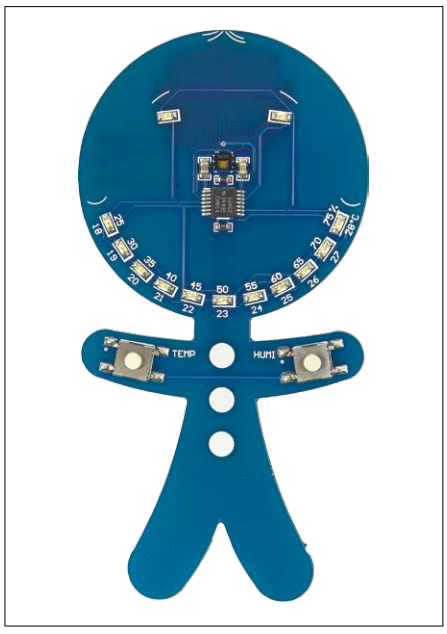


Figure 2. The assembled prototype.

Temi

Temi (the name is a portmanteau of '**te**mperature' and 'hu**mi**dity') is therefore designed both to look attractive (see the images and printed circuit board) and to make the possibilities opened up by modern electronics accessible even to the very young. The circuit board is made to resemble a traditional jumping jack toy [1], and carries a microcontroller, an environment sensor and a sprinkling of LEDs. Replacing the pull-string there are two pushbuttons. Pressing one of the buttons will display either the current temperature or the current humidity using the LEDs: both these functions are provided by the TI type HDC1080 sensor chip [2]. Processing the outputs of this device and driving the display is the job of an MSP430G2402 low-power microcontroller [3], also from TI. The whole circuit is powered by a tiny lithium coin cell, which should last for years.

A look at the rendering of the printed circuit board in **Figure 1** shows the child-friendly component layout: two green LEDs form Temi's eyes, the sensor and the microcontroller make up the nose, and an arc of eleven LEDs forms his broad grin. The buttons for temperature, on the left, and humidity, on the right, are on the (stylized) hands. We think our assembled prototype in **Figure 2** looks rather dapper.

A special circuit

All in all there are 13 LEDs in the circuit. Without some form of multiplexing, this would require 13 I/O lines on the microcontroller. Adding on the requirements for the buttons and for communication with the sensor, we soon come to the point where we will need to resort to a microcontroller with rather a lot of pins. Using a conventional multiplexing configuration for the 13 LEDs will not save us enough pins. For n LEDs at least $2\sqrt{n}$ pins are needed, where $\sqrt{n}$ should be rounded up to the next integer. In our case we therefore need eight pins, because $\sqrt{13} \approx 3.6$, which rounds up to 4. It is possible, however, to use rather fewer pins, using a particularly clever trick called 'charlieplexing'. How this works is explained in the **text box**. Some readers may raise a skeptical eyebrow, but we assure you that we are not pulling your leg: there is even a Wikipedia article on the subject [4]

and, over ten years ago, the technique featured in its own Elektor article [5]. Looking at the circuit in **Figure 3** we can see that we manage to drive twelve LEDs using just four I/O pins. LED2 is the LED left over, and that is driven directly from its own port pin. Another saving in comparison to the conventional arrangement will also be apparent: the LEDs have no series resistors! In practice this works reasonably well as the output drivers of the microcontroller do not have a particularly low impedance. Bearing in mind the LED's threshold voltage and that the supply voltage is only 3 V, we find that the maximum current through an LED is only 20 mA, which it can handle comfortably.

It is also important to note that for the LEDs to be bright enough they must be high-efficiency types: the average current through one of the charlieplexed LEDs is only about 1.6 mA. In our prototype we used LEDs from Würth Elektronik: type 150080GS75000 for the green LEDs, and with the 'G' replaced by 'Y' or 'R' for the yellow and red LEDs respectively.

Unfortunately the microcontroller does not have a built-in I^2C interface to communicate with the sensor. For this reason we therefore need to implement the interface 'manually' in software using bit-banging, taking advantage (as we also do in the case of the pushbuttons) of the selectable pull-up resistors on the microcontroller's inputs. The sensor IC has a very low current consumption, and, for the sake of simplicity, is powered via R1 from an I/O pin on the microcontroller. Together with C1 this resistor forms a lowpass filter that helps suppress interference and ensures a clean supply to the analog circuitry. The arrangement also allows the sensor to be turned on only when a reading is wanted and to be powered down completely in the quiescent state.

Display and operation

Temi's mouth consists of an arc of eleven LEDs. The two LEDs at the extremities of his smile are red, the next two LEDs towards the center are yellow, and the remaining seven LEDs in the middle are green. This arrangement enables us to display relative humidity from 25 % to 75 % in steps of 2.5 %, and temperature from 18 °C to 28 °C in steps of 0.5 °C, which should be adequate for normal indoor use in temperate climes. We use a little trick to get double the resolution that you might expect using the eleven LEDs: a half-step is indicated by lighting two adjacent LEDs rather than one. So, for example, we indicate a temperature of 21.5 °C by lighting the LEDs corresponding to

Charlieplexing

The basic idea behind charlieplexing is to connect two LEDs in antiparallel between each pair of I/O port pins. It must therefore be possible to switch each output to either ground or the positive supply: open-drain or open-collector outputs are not suitable. Using two pins we can drive one pair, or two LEDs; with three pins we can connect three pairs, or six LEDs; with four pins we can use six pairs, or 12 LEDs, and so on.

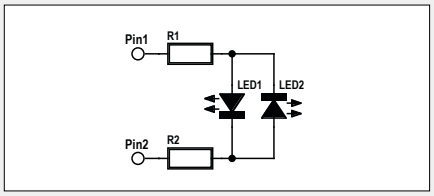
In the two-pin case there is no point in using charlieplexing as we can already drive two LEDs directly. For three or more LEDs the trick only works if the microcontroller has three-state outputs, that is, if the outputs can be set to a high-impedance state. This ensures that the unused outputs do not interfere with the two outputs that are being used to drive the LED in question.

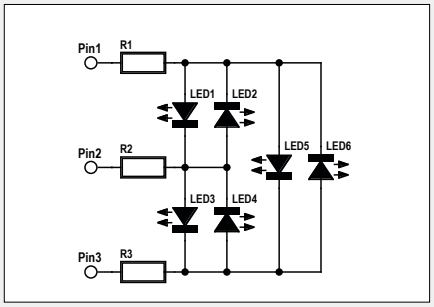
A couple of other points are worth noting. Using direct drive we can light not just one LED, but any combination of them simultaneously. Using normal multiplexing we can light up to $\sqrt{n}$ LEDs at a time, where n is the total number of LEDs: for example, using seven pins and an LED array with four rows of three columns each we can drive 12 LEDs with up to four (all in the same column) lit simultaneously. In this example the LEDs in the three columns would be lit in turn in consecutive time slices; to avoid flickering the column multiplex rate would have to be at least 3 x 30 = 90 Hz.

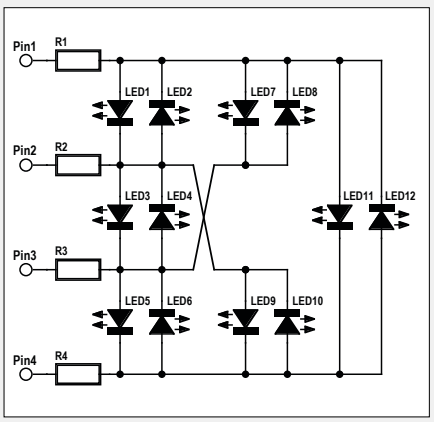
By contrast, in a charlieplexing arrangement, we can only light one LED at a time. This means that the multiplex rate has to be higher: for 12 LEDs being lit simultaneously we must run at a minimum of 12 x 30 = 360 Hz to ensure that flicker is not apparent to the human eye.

But that is not the end of the story: with direct drive each LED can receive its full rated current and so light brightly, but time-slicing the LEDs reduces their brightness as they are only turned on for a fraction of the time. In a normal multiplexing arrangement with, for example, three columns, the LEDs only receive on average one third of their maximum current. In the charlieplexing example that fraction reduces to one twelfth.

A small advantage of charlieplexing is that, whereas implementing normal multiplexing using a microcontroller can involve quite a bit of coding, charlieplexing can be implemented by setting up a table of the output bit patterns required for each LED and simply cycling through them.







both 21 °C and 22 °C.
The LED eyes normally light continuously when Temi is active: only when the battery voltage eventually starts to run low will they start to flash instead. To save power Temi only enters active mode when one of his buttons is pressed. A reading will then be taken (temperature if the left button was pressed, humidity if the right button was pressed) and the result displayed for five seconds. After that Temi will return to his quiescent state.

Battery life

Although in principle it would be possible to have Temi's coin cell provide continuous power for the microcontroller and its LEDs, he will not survive long in such a configuration. Instead Temi is designed to spend most of his life hanging on a wall asleep, in which case the battery will last practically forever. A typical CR2032 lithium cell made by Varta has a claimed usable capacity of around 170 mAh, and Varta guarantees a shelf life of ten years with a self-discharge of 1% per year. We can now make a rough calculation to see how long Temi's battery will last in actual use.
First let us consider the quiescent current consumption of the electronics. The sensor is completely powered down when the processor is asleep, and the processor itself goes into deep sleep mode after five seconds of inactivity. In this mode almost all of the processor's internal modules and clocks are deactivated and it can only be woken up as a result of a reset or a dedicated interrupt (here, when pressing a button causes the level on a specified input pin to change). The result is an extremely low quiescent consumption of only 100 nA. The theoretical battery life in this mode is therefore 170 mAh / 0.1 µA = 1.7 x 10^6 hours, or about 194 years. Even if we take into account leakage currents on the circuit board and through capacitors this result will not change much, and so as long as Temi is not disturbed we will be certain to achieve 10 years of battery life.
When a button is pressed Temi springs into life and the microcontroller, running at 1 MHz, will draw about 320 µA. To this we should add the 1.5 µA drawn by the sensor and up to 20 mA for the LEDs. In total, then, we should reckon on a current draw of about 20.3 mA for the next five seconds. A button press therefore discharges the battery by 5 s x 20.3 mA = 101.5 mAs = 28.2 µAh. So your little darlings can activate Temi 6030 times before you will need to change the battery. That corresponds to four presses a day for a good four years (by which time they will probably be able to change the battery for themselves!).
The microcontroller also measures the battery voltage and flashes Temi's eyes to warn when this falls below 2.8 V: the sensor is only guaranteed to work reliably down to 2.7 V.

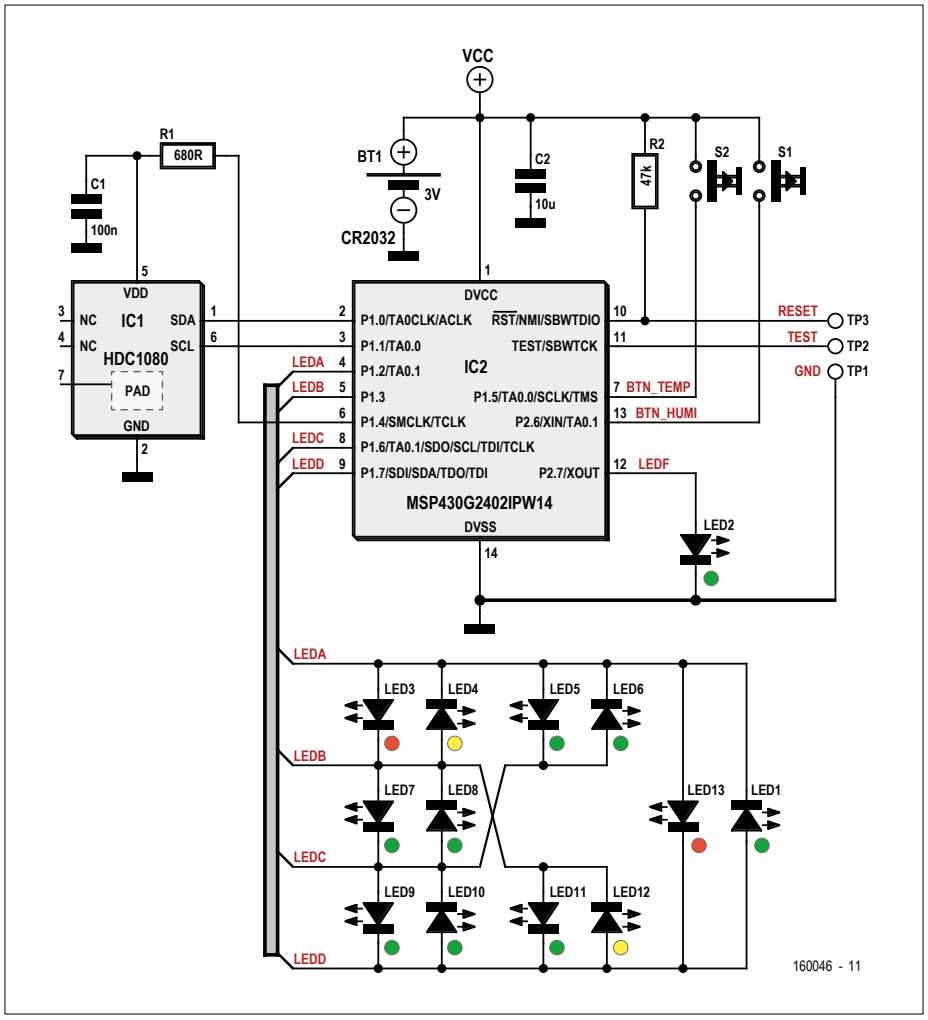


Figure 3. The circuit is reasonably straightforward: the sensor, the microcontroller, and 13 LEDs driven in a charlieplexed arrangement.

Software and all that jazz

The design files for the printed circuit board, the parts list and the software (both as source code and as a hex file that can be programmed directly into the processor) are available for free download from the Elektor web page accompanying this article [6].
The software is fairly straightforward and so only a couple of remarks are necessary. First, a macro is defined to light each LED, to isolate the complexities of the charlieplexing arrangement. Then a table is generated to convert sensor values to LED patterns. Pressing a button triggers a state machine with states STATE_STANDBY (deep sleep mode), STATE_TRIGGER (read battery status and start a measurement) and STATE_DISPLAY (show results on LEDs). The LEDs are driven in the main loop, with the measurement initiated from within the watchdog timer interrupt, which is called every 30 ms. This makes it simple to respect the timing constraints of the sensor, and the state machine can be advanced to its next state on each call. Between interrupts the processor enters sleep mode. The software was written using IAR Embedded Workbench [7].
The author used a standard low-cost

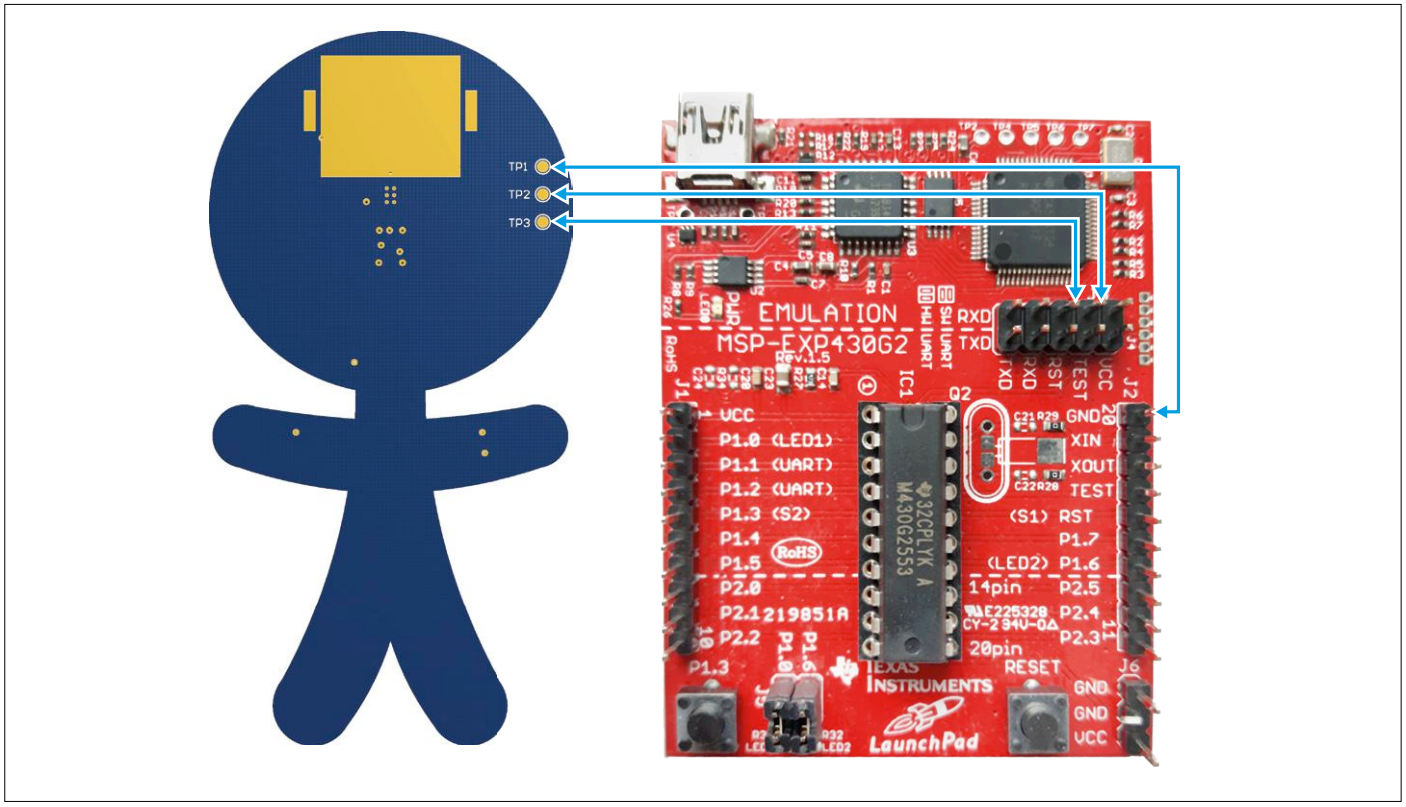


Figure 4. The connections between the Launchpad board and Temi (rear side) to allow his microcontroller to be programmed with the firmware.

MSP430 Launchpad [8] to program the software into the device, removing jumper J3 and connecting the programming signals directly to Temi (see **Figure 4**). This allows Temi's MSP430 to be programmed from the Launchpad using the 'Spy-Bi-Wire' protocol.

(160046)

Web Links

[1] https://en.wikipedia.org/wiki/Jumping_jack_(toy)
[2] Sensor: www.ti.com/product/HDC1080
[3] Microcontroller: www.ti.com/product/MSP430G2402
[4] https://en.wikipedia.org/wiki/Charlieplexing
[5] Elektor article on charlieplexing: www.elektormagazine.com/magazine/elektor-200607/18310
[6] www.elektormagazine.com/160046
[7] www.iar.com/iar-embedded-workbench
[8] www.ti.com/tool/MSP-EXP430G2