to capture every detail and aspect of the slide as faithfully as possible. Needless to say, this type of scan generates a huge file and takes a more than a few minutes to complete. The other type allows you to manipulate the image as it's being scanned and stored, for example to lighten or darken it, crop it, alter the contrast, compress it, hide its scratches, or change its color balance. However, I prefer to make an archival scan and then tweak it with Apple's Aperture, the photo-management software I'm most comfortable with.

Alas, the archival scans were consistently underexposed. The scanner uses a multiple-exposure scheme, in which the image is exposed and scanned twice. A bright exposure wrings detail out of the dark and shadowy parts of the image, and a dimmer exposure makes the most of the brightly lit areas. The SilverFast software then combines the two exposures into a single picture with high dynamic range, which in theory captures good detail in both bright and dark regions of the image. In practice, scan after scan was too dark.

The archival scans also take lots of time and disk space. To make an archival scan at 7200 dpi, and with 48-bit color depth, took 6 minutes and 40 seconds. The resulting TIFF image file occupied 383.1 megabytes.

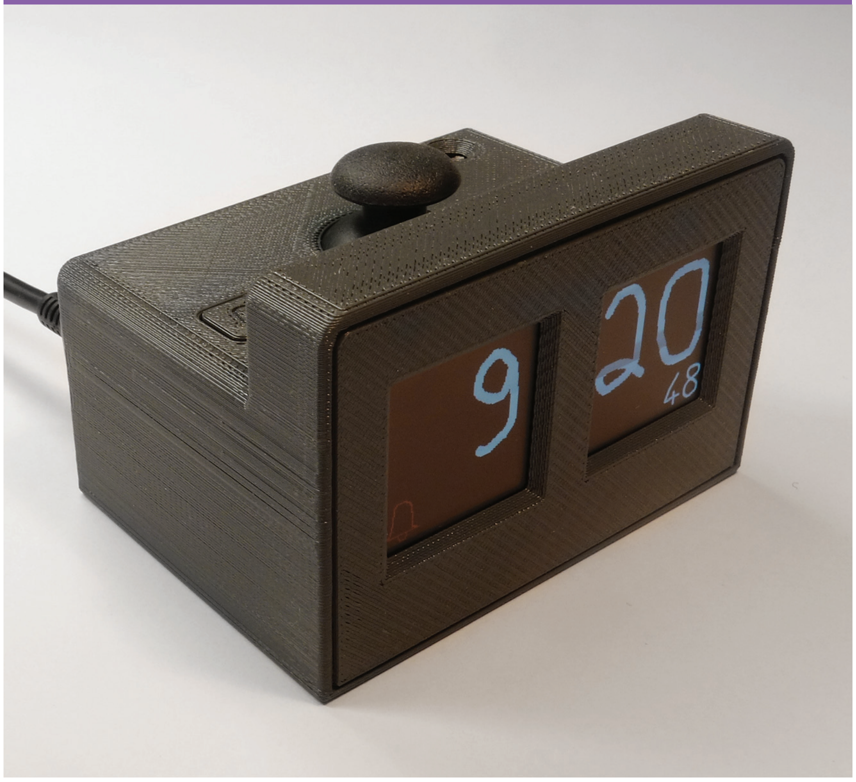
Should the software problems get fixed, the Plustek OpticFilm 8200i Ai would be a tremendous bargain in a shrinking market that could really use one. But as things stand, I can recommend the unit only with a couple of caveats. For those who really want to archive slides, don't want to spend four figures, and won't settle for less than perfection, the wait isn't over. Your move, Plustek.

—GLENN ZORPETTE

Read an extended version of this review online.

TIME FOR GADGETEER

MICROSOFT'S SYSTEM FOR PROTOTYPING MAKES BUILDING YOUR OWN DEVICES EASY



STEVE HODGES

WE LIVE IN A HIGH-TECH WORLD, SURROUNDED BY NEW gadgets like smartphones and Internet TVs, along with other consumer products that were unheard of a decade or two ago. It's not surprising that we often overlook more established electronic devices. Take, for example, the humble alarm clock. Despite digital convergence, many of us still like a dedicated device that lets us know what time it is if we wake in the night and summons us to action in the morning. Sadly, alarm clocks appear to have fallen behind our other home electronics in their sophistication. So I decided to build a better one. ● For this I used Microsoft .NET Gadgeteer, a platform I helped develop as part of my day job in the Sensors and Devices group at Microsoft Research Cambridge, in the United Kingdom. Our work includes the SenseCam used at the heart of Gordon Bell's MyLifeBits project [see "Total Recall," *IEEE Spectrum*, November 2005]. As one of the few groups at Microsoft Research that creates new hardware, we designed Gadgeteer as a rapid prototyping system for our own needs. But we saw so much interest from others that we released it to the general public as an open-source platform in 2011. Several manufacturers now supply Gadgeteer

hardware that works in conjunction with free-to-download software.

So what extra features could anyone possibly want from an alarm clock? For starters, I've always found the typical controls frustrating. And the practically Jurassic seven-segment LCD or LED numeral display ubiquitous today could be much improved.

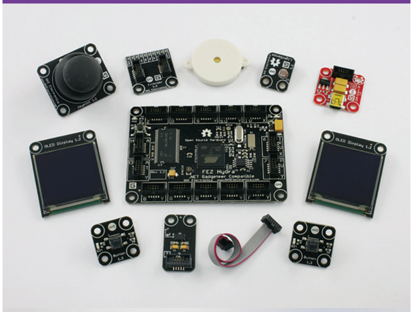
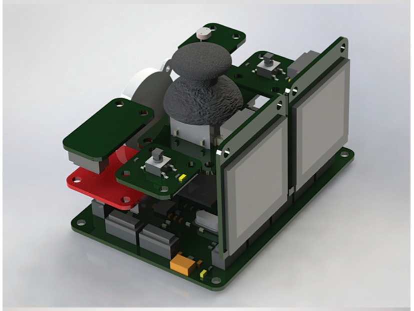
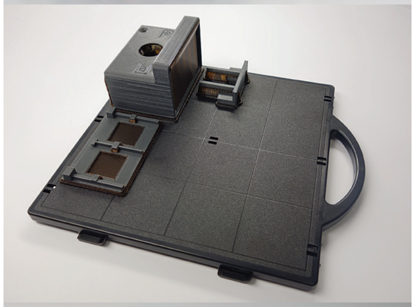
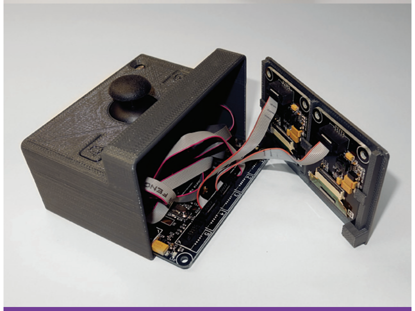
Gadgeteer is founded on a solderless approach that lets you build a device easily—and quickly rebuild it if it doesn't work quite how you imagined. You can readily connect hardware modules with various capabilities, such as a camera or an accelerometer, with pluggable cables. This gives a lot of freedom in designing both the functionality and the form factor of the hardware. About a hundred different Gadgeteer modules are sold online.

I began this project by selecting a thumb-joystick module and a couple of button modules for the controls. In my design, holding down one button selects the clock time and the other selects the alarm time. Pushing the joystick left and right sets the minutes; up and down sets the hours.

When it came to the display, my wife finds the dim glow that LCDs emit annoying at night, even when they are displaying black, so I chose two 128- by 128-pixel OLED screen modules, which don't have the glow problem. This allowed me to replace the blocky digits of yesteryear with bitmaps of hand-drawn numbers. I used a light-sensor module to control how bright the display is and added an accelerometer to silence the alarm with a simple shake of the clock. With Gadgeteer it's also relatively easy to build your own modules; I did just this to make the actual alarm noises by using a cheap piezoelectric sounder connected to an extension module.

For the brains of the alarm clock, I connected all these modules to a Gadgeteer mainboard—in this case a Fez Hydra from GHI Electronics, powered by a 240-megahertz ARM9 processor. Like all Gadgeteer mainboards, this runs the .NET Micro Framework and can be programmed over USB using a free version of Microsoft Visual Studio running on a host computer. The Hydra costs about US $80, while the module prices range from $5 for a button to $29 for an OLED display.

PLUG AND PLAY





Gadgeteer modules [top] provide the hardware functions required for a DIY alarm clock. A CAD software package [screen shot, second from top] was used to design an enclosure, which was printed [second from bottom] using a 3-D printer. The modules were then fitted and connected [bottom].

FROM TOP: STUART TAYLOR; STEVE HODGES; NIC VILLAR; STEVE HODGES

You can write the code that runs on Gadgeteer devices in either C# or Visual Basic. Unusually for a microcontroller system, Gadgeteer is built around an object-oriented and event-based methodology. This means that each physical hardware module is represented in code by a corresponding software object that exposes relevant methods, properties, and events. For example, when a button is pressed, a software event is automatically generated, which in turn triggers a block of code called an event handler to run. This is in contrast to the typical approach of having to program a controller to continually check the button to see if it's been pressed, or writing a traditional interrupt handling routine. Similarly, the accelerometer triggers an event handler to silence the alarm when it detects a shake. The alarm clock display is controlled via a software-only timer object that generates an event once every second.

Finally, if my new alarm clock is to be accepted alongside all the other consumer electronics in my bedroom, it must have a decent case. Handily, there are 3-D models of most Gadgeteer modules available at http://gadgeteer.codeplex.com. You can import these into computer-aided design software such as Solidworks or Autodesk's free-to-use 123D, and then arrange them in virtual space to match the physical hardware. Then you can design the enclosure around them in the CAD software and send the result to a 3-D printer or laser cutter. I used the Dimension Elite 3D Printer we have in our lab, but services like Shapeways or Ponoko can create and mail you an enclosure within a few days.

Now that I have the basic version of my clock working, I'm already thinking of making more upgrades. For example, I'd like to try using the accelerometer as an alternative way to set the time. A ZigBee module would let me display the outside temperature reading from a wireless thermometer. More ambitiously, I could use a Wi-Fi module to connect the clock to my home network and have the clock synchronize with calendar events, perhaps even wish me a happy birthday. Or maybe I'll just prowl around my house looking for other electronic devices to bring into the 21st century. —STEVE HODGES