

.NET Gadgeteer: A Platform for Custom Devices

Nicolas Villar¹, James Scott¹, Steve Hodges¹, Kerry Hammil¹, and Colin Miller²

¹Microsoft Research

²Microsoft Corporation

{nvillar, jws, shodges, khammil, colinmil}@microsoft.com

Abstract. .NET Gadgeteer is a new platform conceived to make it easier to design and build custom electronic devices and systems for a range of ubiquitous and mobile computing scenarios. It consists of three main elements: solder-less modular electronic hardware; object-oriented managed software libraries accessed using a high-level programming language and established development environment; and 3D design and construction tools designed to facilitate a great deal of control over the form factor of the resulting electronic devices. Each of these elements is designed to be accessible to a wide range of people with varying backgrounds and levels of experience and at the same time provide enough flexibility to allow experts to build relatively sophisticated devices and complex systems in less time than they are used to. In this paper we describe the .NET Gadgeteer system in detail for the first time, explaining a number of key design decisions and reporting on its use by new users and experts alike.

1 Introduction

Over the past decade there have been a great many examples in the pervasive and ubiquitous computing research communities where the creation of a new type of digital device has led to valuable research insights. These include mobile battery-powered devices such as SenseCam [18] and Ambient Wood probes [28], situated and augmented displays like the Prayer Companion [11] and Augurscope [29], and distributed systems like the Local Barometer [12] and telepresence proxies [20]. In each of these cases, custom hardware and software in a custom form factor were critical to building devices that could be evaluated in realistic settings.

Of course, developing custom devices from scratch requires a considerable investment of time, money and expertise. Anything which can be done to minimize these requirements, thereby accelerating the process and increasing the scope for exploring a wider design space, is very valuable. A large range of tools and platforms introducing many innovative concepts in this regard have become established over the past few years. For example Phidgets [14] makes it easy to put together new combinations of sensors and actuators in conjunction with a PC, which is valuable in applications with generous size, cost and power consumption requirements. For more constrained applications with basic requirements in terms of processing and I/O bandwidth, platforms like Arduino [1] are an ideal choice. For space-critical and battery-powered distributed sensing applications a range of Mote-like devices [17] are a good option. We review these existing approaches and others in detail in the following section.

Despite the many qualities existing platforms bring collectively, we envisage an integrated platform for the creation of custom devices which simultaneously provides:

- **flexibility** over device **form factor** as well as the **hardware and software**;
- **accessibility** to new users and **extensibility** of the platform;
- **versatility** to scale up to **sophisticated standalone devices** and **complex systems**.

This paper describes in detail for the first time the platform we have created to this end, which we call Microsoft .NET Gadgeteer (abbreviated as Gadgeteer hereafter). By combining the above elements we seek to extend the reach and the sophistication of existing tools. Our research contribution is a platform which enables ourselves and others to more effectively explore and iterate new research concepts and design ideas.

We start this paper with an overview of the large body of related work, which includes both research papers and commercial products. Following this we present an example Gadgeteer device implementation which grounds the subsequent three sections – these detail many of the design decisions we took as the Gadgeteer system developed from a research concept to an open-source project with commercially available components. Although many of these decisions are tightly bound together, we have broadly split them into aspects relating to the electronic hardware, the software experience, and mechanical design and we present them in that order. Finally we report on the various ways in which we have evaluated and applied our platform.

2 Related Work

There is a great deal of previous work in the broad domain of tools to support the design and creation of electronic devices, both in the form of research papers and commercial products. To help set our work in this context, we focus the discussion around the broad areas which we identified above as key properties.

2.1 Accessibility

We believe that the accessibility of a tool is generally an important factor in its adoption. In the embedded development space one system with a particularly low barrier for entry is littleBits, which consists of tiny circuit boards with simple functional elements which snap together magnetically [6]. This results in a system requiring no soldering, wiring or programming. Of course, the ‘pre-programmed’ nature of the system coupled with the relatively simple functionality embodied in each module limits the scope of what can be built, but the system is great for basic ‘hardware sketching’. Like littleBits, with the Lego® Mindstorms system [20] it is quick and easy to assemble and re-configure electronic systems from the 8 different input and output modules available but in this case there is the added flexibility of a central controller unit, programmed using a visual programming language which makes it easy for researchers and kids alike to use.

Arduino [1], targeted at designers, supports a range of embedded development circuit boards and also leverages an accessible programming environment [4]. Devices

are programmed using a derivative of the Wiring language and an integrated development environment (IDE) derived from the Processing IDE. The language is a simplified version of C, which, together with the IDE provides a minimalist and beginner-friendly programming experience. This accessibility has resulted in widespread online documentation in the form of a myriad of design ideas and tutorials, which has in turn made it even easier for others to pick up the platform and extend their knowledge of it. The mBed programmable microcontroller platform [3] is similar in many ways to Arduino, but one key difference is the online IDE, which is particularly accessible because it can be used via a web browser without installing any software.

Like Arduino, the d.tools system [16] was also conceived to support designers with little or no programming experience to prototype functional devices. The d.tools authoring environment provides a visual programming interface, intended to more closely resemble a storyboarding process than a traditional textual programming activity.

Both the MetaCricket toolkit [24], and more recently the xTel platform [35] employ a virtual machine running on an embedded microcontroller, together with a set of high-level libraries to access and program against sensors and actuators, as a mechanism to lower the complexity of programming embedded hardware.

Inspired by littleBits and Mindstorms, in Gadgeteer, we employed a modular hardware design as a way to lower the barrier to entry for users with little hardware experience (see Section 4). Like Arduino, we opted for a textual programming environment with simplified API and built-in helper libraries to aid beginners (see Section 5). We also developed a system of drivers, which allows the underlying implementation of individual hardware modules to be abstracted for beginners.

2.2 Versatility

Another important attribute of a platform is versatility – the ability to use the same platform in different ways. Of course, this is often at odds with accessibility, because the factors which make a platform easy to use often also limit its ability to address complex tasks. However, as others have pointed out [26], the combination of “low threshold” accessibility with the versatility of a “high ceiling” is a valuable goal. A common versatility feature for hardware toolkits is support for integrating additional electronic components. For example, many Arduino boards share a standardized layout of stackable connectors, which provide access to a number of digital, analog and communication interfaces on the processor. Arduino can be augmented either by wiring custom electronics to these connectors, or by using a *shield* – a pre-made extension board. Many hundreds of different shields exist [2], and popular ones include functionality such as Ethernet, GPS and motor control.

A similar approach of baseboards and stackable add-on modules has been used in a number of other embedded platforms, including Gumstix [15], Smart-Its [13] and Berkeley Motes [17]. Another related system is the Bug Labs BUG platform [8], which includes slots into which a small range of hardware peripherals can be attached. However, all these platforms tend to be limited in the number of additional modules which can be connected. For example, the BUGbase unit supports just 2-4 modules and with Arduino complex shields will use up much of the I/O functionality

exposed by the connectors, so in many cases only one shield can be used at a time. In some cases – depending on the number and selection of I/O used by different shields – it is possible to stack and use a number of shields together, but it is not possible to independently chose a range of shields and assume they will all work together.

More recently, the Grove [10] and TinkerKit products [33] have introduced Arduino shields into which a number of hardware modules can be connected. These provide a way to extend Arduino hardware without the need for soldering, for peripheral modules based on a small set of common electrical protocols.

The Gadgeteer design includes a novel and practical system of *sockets* as a way to extend the basic processing unit with additional hardware modules (see Section 4.1). In particular, the use of sockets and multiple *socket types* maximizes the utilization of different processors' I/O capabilities, supports many different combinations of hardware modules and provides a way for different hardware modules to be developed independently of each other while ensuring compatibility between them.

2.3 Flexibility of Form Factor

In our experience, the form factor of a device – its size, shape and finish – is becoming increasingly important, especially when a prototype is to be deployed in a user-facing scenario. A variety of established CAD tools may be used in conjunction with a growing number of 3D printing ‘direct digital manufacturing’ techniques, thereby enabling experts such as industrial designers to turn their concepts into real artefacts. In recent years, online tools and services have started to appear – both in terms of 3D design tools such as the TinkerCAD 3D designer [33] and 3D printing services like Shapeways [32]. We have also started to see applications which greatly simplify the process of 3D design for certain form factors such as MagicBox [23].

Of course, in order to get the most from a custom enclosure, it is important to have control over the form factor of the electronic assembly it contains. As described above, many Arduino implementations are based on a circuit board which supports a number of plug-in shields – resulting in a very specific form factor for the assembled electronics, with little flexibility to readily change this. However, many derivatives and third-party variants of Arduino have been developed to support applications with space constraints or specific form-factor requirements. These include the LilyPad Arduino [7] for wearable computing; the Fio Arduino [9], intended for wireless applications and including a Bluetooth chipset and a rechargeable battery, and the See-duino Film [31], which uses a flexible circuit in place of a rigid PCB.

One approach to supporting flexibility in the form factor of a given device taken by a number of systems is the use of cables to connect physically discrete peripheral modules to a central processing unit. The Phidgets [14] toolkit, for example, supports a collection of sensors and actuators that connect to a PC either directly via USB, or via a USB interface board. d.tools [16] takes a similar approach, allowing the components which comprise the interface of a prototype system to be readily embedded into objects. In the Switcharoo system [4] this approach is taken even further by enabling simple controls to be added to foamcore prototypes in such a way that an RFID reader can detect interaction with them and this information was used in the Macromedia

Director authoring environment. The Calder toolkit [21] built upon this work by further untethering the user interface elements of the prototype from the desktop.

Of course, in the above cases a PC of some kind is required, although with the latest integration technologies such a device can be very compact, e.g. the Phidgets single board computer. In the case of LEGO Mindstorms, the controller ‘brick’ can more easily be integrated into a device and flexibly connected with sensors and motors via cables, although the platform is mainly aimed at robotics applications where the appearance of the resulting construction is less important than its physical configuration.

One of the most refined platforms in terms of the appearance of the resulting devices is the Bug Labs system [8]. BUG modules and the BUGbase are supplied in enclosures which give prototypes a professional appearance. Unfortunately, there is no control over the physical configuration of a completed prototype, other than which module is connected to which socket, so the result is more of a configurable device rather than a free-form prototype.

With Gadgeteer, compact scale of the electronics and connectors, together with the use of flexible ribbon cables to connect hardware modules provides considerable freedom in the design of devices with varied form-factors. We believe that the integration with CAD and rapid manufacturing technologies, see Section 6.2, is also novel.

2.4 Complex Devices and Systems That Are Easy to Build

As outlined in the introduction, our ultimate aim is to enable researchers to build sophisticated devices and systems which empower them to take their work in new directions. We feel that accessibility, versatility and flexibility are key elements in this, and a platform which combines elements of all these is likely to go a long way towards our goal.

In the remainder of this paper we present the .NET Gadgeteer platform. It shares many characteristics with the work discussed above, but we have strived in particular to supplement hardware and software design elements with a high level of control over form-factor. We have also emphasized the extensibility of Gadgeteer along with the power of the underlying software tools. We believe that this unique combination is valuable in supporting the rapid creation of sophisticated standalone devices and complex systems.

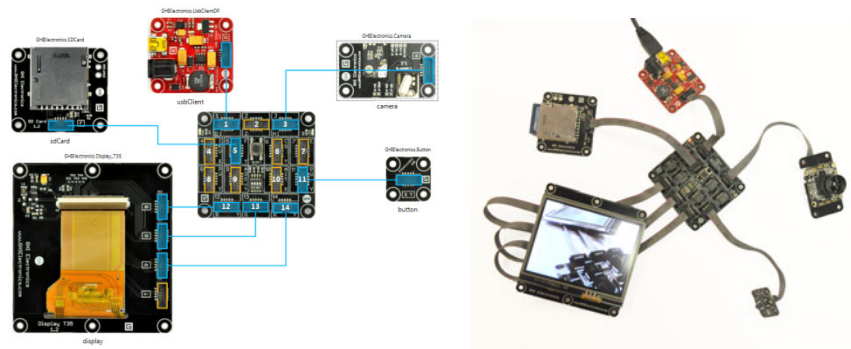


Fig. 1. Designer view (left) and assembled electronics (right) for digital camera example

3 An Example Gadgeteer Device: A Digital Camera

Before presenting details of the design and implementation of Gadgeteer, we illustrate the end user experience for a device which is relatively simple to build using the platform – a digital stills camera. To accomplish this, a Gadgeteer *mainboard* is combined with five *modules*: camera, push button, display, SD card and a USB client module for powering and programming the device. Having created a new Gadgeteer project in Visual Studio, which brings up the Designer view (Fig. 1 left), representations of the hardware modules listed above are dragged and dropped into place. The Designer suggests how to connect modules to compatible mainboard sockets, and the real hardware is then connected accordingly using 7 cables, Fig. 1 right (in this case the display module requires 3 cables).

```

void ProgramStarted()
{
    // Associate events with event-handling methods
    button.ButtonPressed += new Button.ButtonEventHandler(button_ButtonPressed);
    camera.PictureCaptured += new Camera.PictureCapturedEventHandler(camera_PictureCaptured);
}

void button_ButtonPressed(Button sender, Button.ButtonState state)
{
    camera.TakePicture();
}

void camera_PictureCaptured(Camera sender, GT.Picture picture)
{
    // Show the picture on the display
    display.SimpleGraphics.DisplayImage(picture, 0, 0);

    // Save the picture to the SD card
    sdCard.GetStorageDevice().WriteFile("picture.bmp", picture.PictureData);
}

```

Fig. 2. Code listing for digital camera example

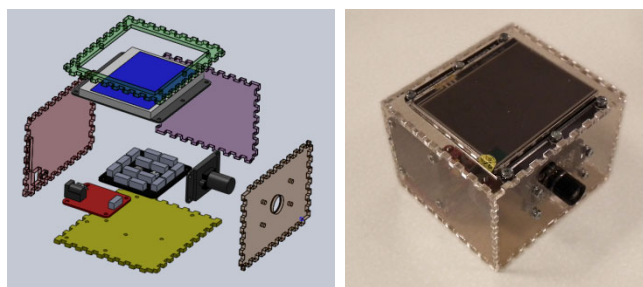


Fig. 3. 3D model of case (left) and realized device (right)

To implement the software logic for the digital camera, two *event handlers* for the `button.ButtonPressed` and `camera.PictureCaptured` events are required, and need to be populated with one and two lines of code respectively (Fig. 2). The device can then immediately be tested. A solid modeling package can readily be used to create a simple casing based on a “jigsaw box” template, automatically using 3D models of the

electronic components as reference geometry to quickly create holes suitable for mounting the modules, exposing the camera lens and so on (Fig. 3). The physical enclosure can be created from a sheet of acrylic or plywood using a laser cutter and the whole thing can be assembled within an hour or so.

4 Modular Electronics

In the example above, we already introduced the notion that .NET Gadgeteer devices are based on a mainboard, to which a number of peripheral modules can be connected. The mainboard includes a processor, memory and a number of physically identical 10-pin sockets. Each module provides certain functionality, such as sensing, actuation, display or communications ability. In this way, designs based on new combinations of electronic components may be rapidly prototyped and iterated upon.

4.1 Socket Types

Key to the design of Gadgeteer is that each mainboard socket supports one or more *socket types*. This is an important concept which we arrived at after some experimentation. Although the sockets are physically identical, which means that only one type of cable is required, each socket can be associated with one more multiple *socket types*, which determine its electrical configuration. We define socket types for many of the common embedded interface standards such as UART serial communications, I2C, SPI, general-purpose digital input and output (GPIO), PWM output, analog input/output, USB host, USB client, an LCD controller interface, and others. The full specification is online at <http://gadgeteer.codeplex.com/>

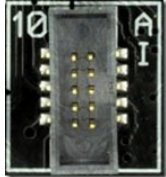
We chose to support a range of interface standards instead of using a single bus type for two related reasons. Firstly, embedded processors typically have hardware support for many of these standards. Secondly, many existing peripheral electronic components already support one (or more) of these standards, but the standard differs from one component to the next. For example, a temperature sensor might have I2C support, while a small OLED display might support SPI. Thus, by supporting many different socket types we enable a wide range of Gadgeteer modules to be built simply with existing components, without requiring an additional processor to perform protocol translation “glue”. This promotes extensibility and versatility as it is easy to build new types of module, and easy to create mainboards based on new processors by simply mapping the pins/functions available on that mainboard to a set of sockets. We expect the set of socket types to increase gradually as new embedded interface standards achieve prevalence.

A key aspect of our implementation is that mainboard sockets can support multiple types. To give a simplified example, the socket type specification for type **A** is as follows: Pins 1, 2 and 10 are connected to the +3.3V, +5V and Ground lines respectively, which is standard across all socket types; in addition pins 3, 4 and 5 are analog inputs (AIN). Furthermore, pins 3, 4 and 6 double as general-purpose input/outputs

(GPIOs). The definition for socket type **I** specifies that pins 8 and 9 are the dedicated I2C data (SDA) and clock (SCL) lines, and that pins 3 and 6 are GPIOs.

A socket on a mainboard can therefore support both socket type **A** and socket type **I** at the same time. The table below shows the actual pin-out and the image to the right shows how the physical connector would be labeled. The number 10 is a unique numerical identifier for that socket on the mainboard. Note that some combinations of socket types are not possible due to clashes between pin allocations.

Pin 1	Pin 2	Pin 3	Pin 4	Pin 5	Pin 6	Pin 7	Pin 8	Pin 9	Pin 10
+3.3V	+5.0V	AIN / GPIO	AIN / GPIO	AIN	GPIO		SDA	SCL	GND



Most modules need just a single socket to connect to a mainboard, although they can use several if necessary. For example, LCD display modules need three sockets due to the large number of signals needed to drive them. To help determine electrical compatibility, modules are labeled with the compatible socket types. For example, a sensor module that needs an I2C connection to the mainboard would be labeled with the letter **I**.

Early on in the design of Gadgeteer, we settled on the use of a common type of connector across the range of different interface standards we support. This is different to many earlier systems, such as Calder [21] but we believe it has two important advantages: firstly, it prevents complication when two or more different interface types are combined on the same physical connector as described above, and secondly it means that only one type of cable is necessary. It is important to note that we have carefully specified the socket types so that connecting incompatible socket types together does not cause the hardware to fail permanently. Thus, while such misconnection is one of the more common errors made when using Gadgeteer, it has temporary effects only. The Designer view (Fig. 1) automatically checks that the connections are compatible, so debugging such misconnections involves double-checking that the physical hardware matches the Designer view.

4.2 Mainboard Examples

We have designed the Gadgeteer socket types to be independent of the underlying processor, enabling a range of different mainboards to be built. The main requirement for the processor is simply enough speed and memory (RAM and flash) to support the .NET Micro Framework coupled with the .NET Gadgeteer core libraries detailed in the next section. We built two prototype mainboards with different processors in-house to demonstrate the processor-independent nature of the platform. The first is based on GHI Electronics' EMX processor module with a 72MHz ARM 7 CPU, 4.5MB Flash and 16MB RAM. It has 14 sockets with 35 socket type letters between them. The second is based on Device Solutions' Nano processor module which includes a 200MHz ARM 9 processor with 8MB Flash and 8MB RAM, with 10 sockets. Commercial retailers have since made their own retail versions based on the same two processors, as well as a third based on a 240MHz ARM 9 processor. See Fig. 4.

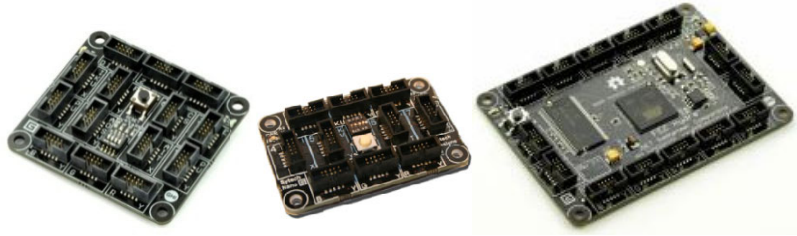


Fig. 4. Example Gadgeteer mainboards, the smallest of which is 37mm x 57mm

4.3 Module Examples

At present over 40 .NET Gadgeteer-compatible modules are either in development or available for public release from retail manufacturers (Fig. 5). A few examples are:

- A **button+LED** module using socket type X (3 GPIOs, 1 interruptable). The button is wired to a digital interrupt input while the LED is wired to a digital output; there is no processing onboard.
- An **accelerometer** based on the Bosch BMA180 sensor (with no additional processing elements) which uses socket type I (I2C interface).
- A 320x240 **LCD display** module using socket types R, G and B. Mainboards can typically gang these socket types up with type Y (GPIO), so, for “headless” devices not using an LCD, the mainboard sockets are still useful.

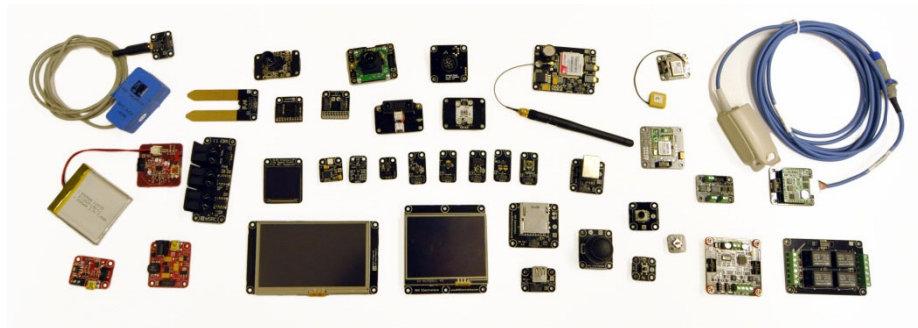


Fig. 5. Gadgeteer compatible modules include displays of various sizes & resolutions, a variety of sensors, actuators, user controls, storage and communications modules, plus power supplies

4.4 Supplying Power

Every Gadgeteer socket type provides access to common 5V and 3.3V power buses and ground through three dedicated pins. Modules are typically powered from these pins, but it is the responsibility of the user to ensure that exactly one module is *providing* power. Suitable modules are colored red. The most often used red module provides USB Client functionality, and can be powered from the USB port of a PC or

from a mains adapter. Other red modules that have been prototyped support a range of Li-ion rechargeable batteries and a USB serial connection (see Fig. 5 bottom left).

5 Object-Oriented Software Architecture

Gadgeteer software is written in C# using Visual Studio (or the free version Visual C# 2010 Express), and uses the .NET Micro Framework (NETMF) platform for the underlying runtime support. This combination delivers a fully-featured development environment, with code auto-completion, in-line help, and the ability to debug code “live” with breakpoints and stepped execution, using “managed” C# code (i.e. strongly typed, object oriented and memory safe code).

On top of NETMF, we provide a set of software libraries known as GadgeteerCore, also written in C#, and a Designer plug-in for Visual Studio. These provide a framework which enables end user code to be abstracted away from particular mainboards and supports a wide and extensible range of modules, while allowing mainboard and module designers to provide device-driver-like supporting code for their hardware.

GadgeteerCore’s API is event-driven rather than using a “while(true)” loop common in many microcontroller scenarios. This promotes versatility by more easily supporting complex behaviors where multiple simultaneous control loops are occurring. End user code starts with the ProgramStarted() method called at startup, in which event handlers or timers are set up, as shown in Fig. 2.

5.1 Supporting Development of Module and Mainboard Drivers

One of the key elements of Gadgeteer is that mainboard and module hardware must come with software “drivers”. This simplifies the end user experience; an end user of a camera module is not exposed to the underlying protocols by which the camera is controlled, but instead uses a Camera object with an API such as TakePicture() method and PictureCaptured event.

However, places an additional burden on hardware developers who may not have much experience with C# or NETMF. We therefore tried to make this as simple as possible through providing templates for modules and mainboards. Each consists of:

- A readme.txt with instructions on how to customize the other files
- A GadgeteerHardware.XML file, which instructs the Designer on how to show the module/mainboard to the end user and on socket compatibility issues
- A C# software file containing an example driver to be customized
- A WiX-based utility (<http://wix.sourceforge.net/>) which automatically produces an “MSI” installer for distributing drivers to users as part of the compilation process.

5.2 Module Examples

Revisiting the three example modules from Section 4.3:

The **button+LED** module comprises very simple hardware, but the software provides a surprisingly rich API. In addition to using the button and LED as two separate

devices, another mode of use is where the button and LED together become a “toggle” where a button press automatically toggles the LED and the user’s code can handle toggle-on and toggle-off events.

The **accelerometer** module hides the I2C interface details (e.g. the device address) and provide a high level API including a property for the high-acceleration detection threshold, an Acceleration class used to return typed 3D acceleration values, and the ability to ask for continuous measurements (at a configurable time interval) – these are implemented using a timer internal to the module software, but the end user simply sets up an event handler which is periodically called with the current Acceleration.

The **LCD display** module driver derives from an abstract class DisplayModule in GadgeteerCore. The abstract class provides APIs for end users for outputting to the display using either a windowing toolkit (WPF) or simple bitmap operations. Therefore, the module driver itself simply has to specify the implementation details of how to get bitmaps onto that particular display type (e.g. sending the bitmap to the processor’s built-in LCD controller, or sending the bitmap over SPI).

6 Flexibility of Form Factor

As commented earlier, in many situations the form factor of a device is as important as the functionality it embodies. This is particularly true in research contexts, where enabling user evaluation of a new device or concept with a realistic prototype may be critical. For this reason, Gadgeteer has been designed from the outset to be both compact and to support a wide variety of form factors. Features enabling a number of mechanical construction approaches and the latest rapid fabrication processes have been included in Gadgeteer in a way which we believe sets it apart from most of the previous work described in Section 2.

6.1 Flexibility in the Mounting of Modules

The most obvious mechanical feature of Gadgeteer is the use of ribbon cables instead of rigid PCB-mounted headers or single wires. This permits components to be mounted arbitrarily with respect to one another while using a minimal number of cables. Furthermore, high-density ribbon cables of multiple lengths terminated with IDC connectors are used to reduce the space required. Compared to the industry standard 2.54mm pitch interconnect, Gadgeteer’s 1.27mm pitch connectors require around 25% of the footprint and under half the height.

The mounting holes are another key element of all mainboards and modules. We use a standard 3.2mm diameter hole size which is compatible with both metric M3 and imperial 4-40 machine screws. We maintain a keep-out area around the screw hole which allows the use of mounting pillars which are large enough to incorporate off-the-shelf threaded inserts. The mounting holes are on a fixed 5mm pitch grid which means they can be readily attached to a perforated baseplate. To this end we have designed a push-fit plastic pillar. This pillar is end-stackable, thereby allowing modules to be stacked on top of each other if required.

6.2 Full 3D CAD Integration for More Sophisticated Physical Design

We have created detailed 3D CAD models of each mainboard and module. When modeling an enclosure for a particular device, these may be imported into the CAD tool and used as reference geometry to ensure that mounting holes, apertures and other features align as required. In many CAD packages it is also possible to check for potential interference between adjacent elements of an assembly before a design is realized. We have used this technique with both 3D-printed and laser-cut enclosures.

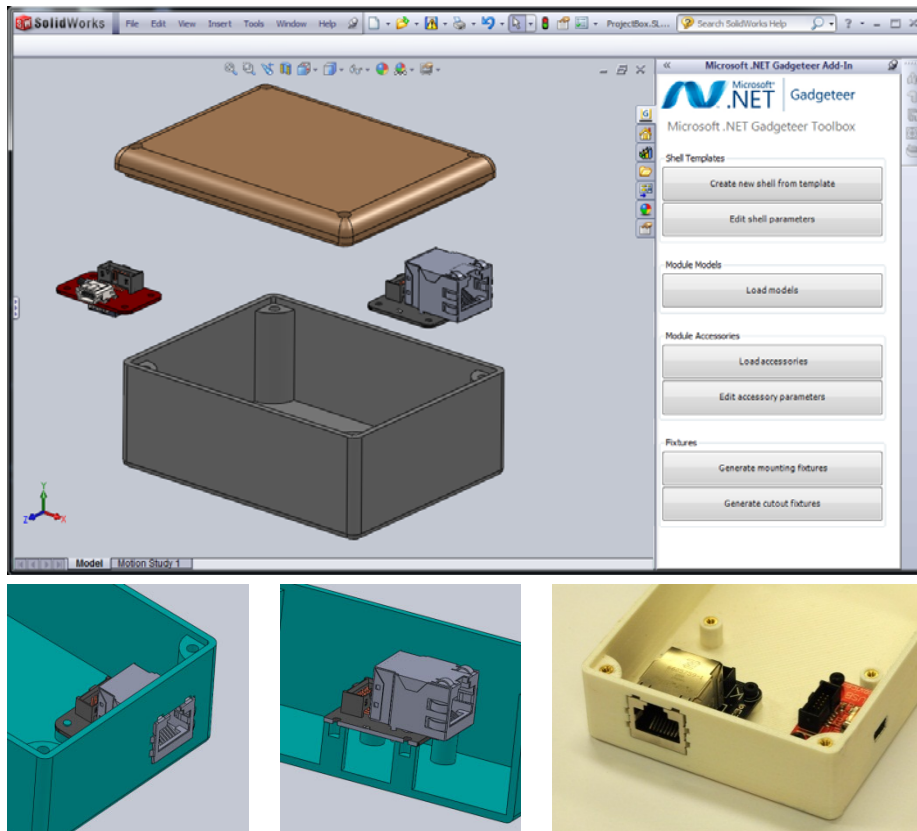


Fig. 6. The Gadgeteer SolidWorks add-in allows users to create simple custom enclosures based on templates and import 3D CAD models of mainboards and modules (top). The add-in also supports automatic creation of the appropriate mounting bushes and apertures (bottom left, middle), allowing the hardware to fit neatly in the 3D printed custom enclosure (bottom right).

We have also developed a plug-in utility for the SolidWorks parametric 3D CAD package which allows a user to specify which mounting holes and apertures are to be used for each module in order that they are automatically incorporated in the design, see Fig. 6. This has proven invaluable because it not only automates the generation of such features, but it results in their automatic re-generation whenever elements of the

design are altered – for example, if a particular module is moved the associated mounting bushes will automatically move to follow. Our plug-in currently supports both laser-cutting and 3D printing enclosure fabrication; the user selects between these to control the type of mounting bush feature generated – either a clearance hole for a plastic pillar to be screwed in place or an extruded bush for a threaded insert.

7 Examples of Gadgeteer in Use

At various points during the development of the Gadgeteer platform, we validated our design and implementation decisions through public workshops. These introduced newcomers to the technology and allowed us to observe and reflect on its utility as a prototyping tool. In this section we reflect on the lessons learned during this process. We have also used Gadgeteer extensively to support our own research projects and we report some of these experiences, using them as a vehicle to highlight what we consider to be the salient properties of the platform.

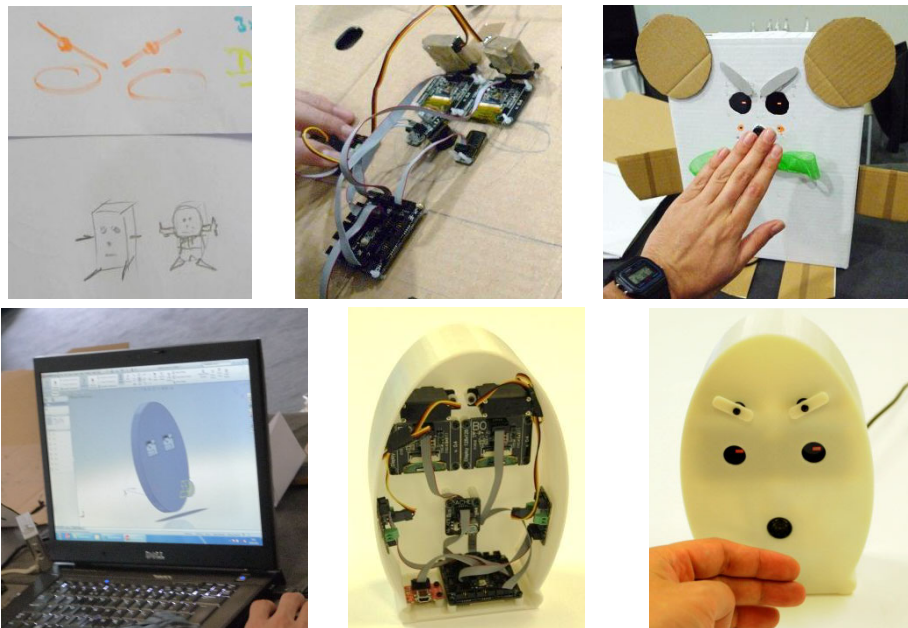


Fig. 7. A day's worth of prototyping – from sketch (top left) to finished device (bottom right), via cardboard prototype and custom 3D-printed enclosure design. The team that created this toy bear had not used C# or SolidWorks extensively before the workshop.

7.1 Accessibility to New Users

We organized a Gadgeteer workshop as part of the TEI'11 conference, which attracts a large number of participants that regularly build custom devices. The workshop

hosted 18 participants of varied backgrounds and skill sets, ranging from interaction design to computer science. We asked participants to form into groups of 3 or 4 members, so that each group had members with programming and design / 3D CAD modeling experience. (NB they had not necessarily used C# or SolidWorks.) The morning was focused on introducing participants to the hardware and software concepts and the SolidWorks CAD support. Participants were then guided through the process of building a simple device – a digital camera, which included a display, shutter button and an additional component of their choosing. The rest of the day was spent in freeform ideation and creation of novel prototypes. Groups were given simple prototyping materials for mounting the hardware modules such as cardboard, tape and plastic screws.

All groups managed to create functional prototypes by the end of the day. One group created a remote control car, complete with a Zigbee wireless controller and animated light effects. Another group created a tool for color-blind people, which would display the name of the primary color of an object held up to its sensor. A third group created an interactive toy bear, featuring a friendly expression which would turn fierce when approached (see Fig. 7). Two of the groups used the 3D CAD tools ‘in anger’ to design robust enclosures for their prototypes. The results were rendered using a 3D printer during the course of the following 24 hours, allowing the assembled and fully-functional prototype to be showcased at the conference’s main demo session, which took place 48 hours after the workshop.

7.2 Hi-fidelity Custom Devices to Support Studies in the Wild

During the development of Gadgeteer, we have used the platform several times in collaborative research projects. In [26] we contributed to a project examining the role of audio clips as “sonic mementos,” which could be captured and subsequently re-experienced by families in a similar way to physical mementos of a holiday. The technical artifact that facilitated this research was a device that looked like an old transistor radio, but was actually a highly custom system for browsing and playing digital audio files, shown in Fig. 8. The device was made by repurposing a genuine radio using Gadgeteer components. The custom design of the device was central to the research questions being addressed, and the ability to deploy the self-contained prototype with participants in their homes was fundamental to the research methodology.

In a second collaboration [19], Gadgeteer was used to develop a smart shopping-trolley accessory: a battery-powered device which was clipped to the handle of the trolley to reveal information about products that were identified using a built-in barcode scanner. This project provided an opportunity to illustrate the value of a high-fidelity, standalone and self-contained device built with Gadgeteer. The resulting device (Fig. 8) includes the barcode scanner, LED display, battery power supply, data logging capabilities, as well as a database of product barcodes that allows it to be used in a real supermarket. The device was iteratively designed and rapidly manufactured over a period of two weeks. Experimental subjects were able to use the device without help from the researchers, who were instead able to focus on observing the interaction and decision-making process. In this sense, we believe that Gadgeteer can help custom research devices such as this ‘disappear,’ and allow the researcher’s energy to focus on the research itself, rather than on managing the technology.

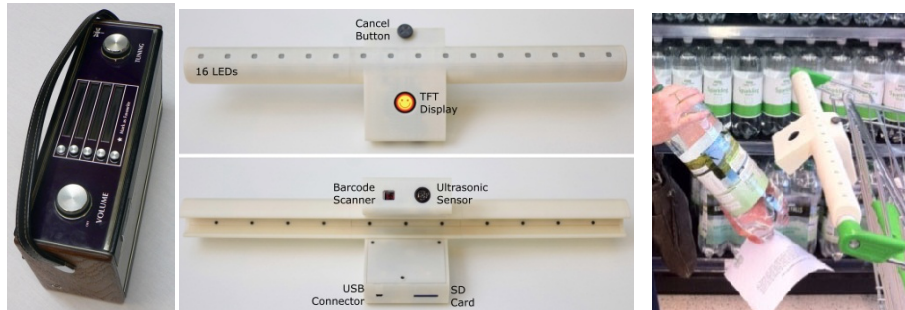


Fig. 8. Two example custom devices built with Gadgeteer to support research deployments: the Family Memory Radio (left) and the Lambent Shopping Trolley handle (middle, right)

7.3 Iterative Design of More Complex Systems

In previously published work [30] we used Gadgeteer to both iteratively prototype and batch produce a number of custom embedded interactive devices which make up a distributed system. In this case, the research programme called for an embedded device that was capable of: sensing the temperature and humidity of the environment; detecting user movement in front of the device; measuring the ambient light levels in the room; presenting a basic user interface; logging data locally, and synchronizing and communicating over a wireless mesh network. As is often the case, there were a number of commercial products that approximated the requirements, but nothing off-the-shelf that fitted the bill exactly.

Using Gadgeteer, we were able to rapidly and iteratively design a solution. Fig. 9 shows four different prototypes, built over a period of four days. The first prototype was a rough ‘hardware sketch,’ built using laser-cut cardboard and used primarily to evaluate the use of a passive infrared sensor to detect user motion. The subsequent prototypes were rendered in plastic using a 3D printer. In a classic demonstration of the value of iterative prototyping, each prototype taught us something valuable, and we were able to feed the lessons into the next iteration. One key aspect was the positioning of the temperature sensor: in the first two examples, the temperature sensor is built into the enclosure, and is only exposed to the outside via a small slit on the front of the case. We quickly found that heat generated by the power supply and the processor was affecting the sensor readings. In the third prototype, the sensor was moved outside of the enclosure. This was found to be still too close to the source of heat, so in the final prototype it was mounted on an external ‘arm,’ which successfully solved the issue. Other changes included the introduction of a rotatory encoder to support user input (between prototypes 2 and 3), the repositioning of the rotary encoder from the top to the side of the device, and the repositioning of the light level sensor from the front to the top of the device. These choices were all affected by functional and usability reasons. This strengthens our conviction that having freedom in the physical form-factor and external design of a device can be as important as having control over its hardware configuration and software it executes.

In order to carry out the research reported in [30], we needed around 40 “copies” of this custom device. We considered designing a custom circuit board and enclosure for this, but decided against it when we estimated the cost of such an effort compared with the material costs of using Gadgeteer components and additional 3D printed enclosures. The latter turned out to be far more cost-effective even before we considered the possibility of re-using the Gadgeteer modules at the end of the project.

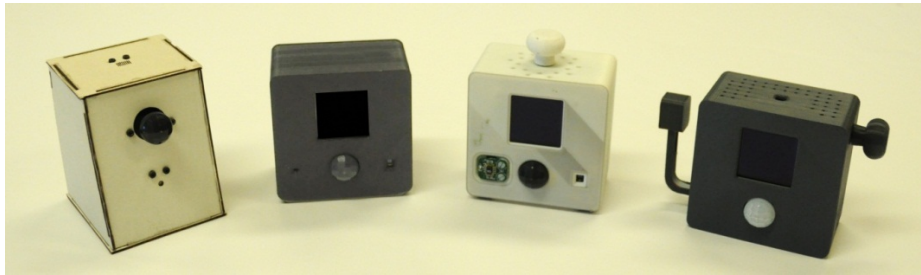


Fig. 9. Iterative prototypes of a custom-built device from initial cardboard prototype (left) through to final design (right) which was used in the batch-production of 40 units

8 Conclusions and Future Work

This paper presented the .NET Gadgeteer system, an integrated platform to support the development of custom devices such as those often used in pervasive and ubiquitous computing research. Our work was motivated by our perception that there was scope to enhance existing tools and approaches and we have described the opportunities we saw in this regard and how our chosen solutions unfolded as the Gadgeteer platform developed. We hope that our experiences will inform future research in this area and encourage others to help us refine and enhance the extensible platform we have created. Although we haven’t conducted a formal evaluation of Gadgeteer in use or a direct comparison with other tools, we also hope that we have demonstrated the value of Gadgeteer by way of examples presented in this paper, and we encourage others to evaluate it and potentially adopt it in their research. To this end we have made the .NET Gadgeteer software freely available for download at <http://gadgeteer.codeplex.com/> under an Apache 2.0 open source licence and we have been working with various manufacturers to create an ecosystem of mainboards and modules which are available to buy – see <http://netmf.com/gadgeteer> for more details.

We are continuing to develop many different elements of Gadgeteer. We are exploring the possibility of supporting languages other than C# and use of an in-browser IDE to extend accessibility of the tool. We are developing an advanced networking library which allows integration of Gadgeteer prototypes with phone-, desktop- and cloud-based applications. And we want to provide even more accessible 3D design tools which integrate tightly with on-line fabrication services to make this aspect of the platform accessible to those who don’t have traditional 3D CAD training and direct access to laser cutters and 3D printers.

Whilst our motivation for developing the .NET Gadgeteer platform was to expedite the process of turning design ideas in the pervasive and ubiquitous computing space into working prototypes, we have also received a great deal of interest from both educators and hobbyists. In the case of education, Gadgeteer appeals because it enables students – whether at high school or university level – to build and program tangible devices using modern programming paradigms and tools, thereby exciting them to learn eminently transferrable computer science skills. We have already started some initial pilot studies in high schools in the UK and US to get experience with Gadgeteer in this domain. In the case of hobbyists the attraction appears to be the simple ability to build sophisticated gadgets with little previous experience remarkably quickly. We are working with several manufacturers to look for opportunities to reduce the cost of Gadgeteer components, in particular the entry cost for a mainboard and minimal set of modules. We believe this will be important to extend accessibility of the platform into schools and for hobbyists.

References

1. Arduino, <http://arduino.cc/>
2. Arduino Shields, <http://shieldlist.org/>
3. The ARM MBed, <http://mbed.org/>
4. Avrahami, D., Hudson, S.E.: Forming interactivity: a tool for rapid prototyping of physical interactive products. In: Proceedings of the 4th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques (DIS 2002), pp. 141–146 (2002)
5. Banzi, M.: Getting Started with Arduino. O’Reilly (2008) ISBN: 978-0-596-15551-3
6. Bdeir, A.: Electronics as material: littleBits. In: Proceedings of the 3rd International Conference on Tangible and Embedded Interaction (TEI 2009), pp. 397–400 (2009)
7. Buechley, L., Eisenberg, M., Catchen, J., Crockett, A.: The LilyPad Arduino: Using Computational Textiles to Investigate Engagement, Aesthetics, and Diversity in Computer Science Education. In: CHI 2008, pp. 423–432 (2008)
8. Bug Labs, <http://www.buglabs.net/>
9. Funnel Arduino, <http://funnel.cc/>
10. Grove System, http://www.seeedstudio.com/wiki/GROVE_System
11. Gaver, W., Blythe, M., Boucher, A., Jarvis, N., Bowers, J., Wright, P.: The prayer companion: openness and specificity, materiality and spirituality. In: Proceedings CHI 2010, pp. 2055–2064 (2010)
12. Gaver, W., Boucher, A., Law, A., Pennington, S., Bowers, J., Beaver, J., Humble, J., Kerridge, T., Villar, N., Wilkie, A.: Threshold devices: looking out from the home. In: CHI 2008, pp. 1429–1438 (2008)
13. Gellersen, H., Kortuem, G., Beigl, M., Schmidt, A.: Physical Prototyping With Smart-Its. IEEE Pervasive Computing 3(3), 74–82 (2004)
14. Greenberg, S., Fitchett, C.: Phidgets: easy development of physical interfaces through physical widgets. In: Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology (UIST 2001), pp. 209–218 (2001)
15. Gumstix, <http://www.gumstix.com/>

16. Hartmann, B., Klemmer, S.R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, A., Gee, J.: Reflective physical prototyping through integrated design, test, and analysis. In: Proceedings of UIST 2006 (October 2006)
17. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System architecture directions for network sensors. In: ASPLOS 2000 (2000)
18. Hodges, S., Williams, L., Berry, E., Izadi, S., Srinivasan, J., Butler, A., Smyth, G., Kapur, N., Wood, K.: SenseCam: A Retrospective Memory Aid. In: Dourish, P., Friday, A. (eds.) UbiComp 2006. LNCS, vol. 4206, pp. 177–193. Springer, Heidelberg (2006)
19. Kalnikaitė, V., Rogers, Y., Bird, J., Villar, N., Bachour, K., Payne, S., Todd, P.M., Schöning, J., Krüger, A., Kreitmayer, S.: How to Nudge In Situ: Designing Lambert Devices to Deliver Salience Information in Supermarkets. In: Proceedings of UbiComp 2011 (2011)
20. Kuzuoka, H., Greenberg, S.: Mediating awareness and communication through digital but physical surrogates. In: CHI 1999 Extended Abstracts, pp. 11–12 (1999)
21. Lee, J.C., Avrahami, D., Hudson, S.E., Forlizzi, J., Dietz, P.H., Leigh, D.: The Calder toolkit: wired and wireless components for rapidly prototyping interactive devices. In: Proceedings of DIS 2004, pp. 167–175 (2004)
22. LEGO Mindstorms, <http://mindstorms.lego.com>
23. Magic Box, <http://magic-box.org/>
24. Martin, F., Mikhak, B., Silverman, B.: MetaCricket: a designer's kit for making computational devices. *IBM Syst. J.* 39(3-4), 795–815 (2000)
25. The Microsoft .NET Micro Framework, <http://netmf.com/>
26. Myers, B.A., Hudson, S.E., Pausch, R.: Past, Present and Future of User Interface Software Tools. *ACM Transactions on Computer Human Interaction* 7(1), 3–28 (2000)
27. Petrelli, D., Villar, N., Kalnikaitė, V., Dib, L., Whittaker, S.: FM radio: family interplay with sonic mementos. In: Proceedings of CHI 2010 (2010)
28. Rogers, Y., Price, S., Fitzpatrick, G., Fleck, R., Harris, E., Smith, H., Randell, C., Muller, H., O'Malley, C., Stanton, D., Thompson, M., Weal, M.: Ambient wood: designing new forms of digital augmentation for learning outdoors. In: Proceedings of the 2004 Conference on Interaction Design and Children: Building a Community, IDC 2004 (2004)
29. Schnädelbach, H., Koleva, B., Flintham, M., Fraser, M., Izadi, S., Chandler, P., Foster, M., Benford, S., Greenhalgh, C., Rodden, T.: The Augurscope: a mixed reality interface for outdoors. In: Proceedings of CHI 2002, pp. 9–16 (2002)
30. Scott, J., Bernheim Brush, A.J., Krumm, J., Meyers, B., Hazas, M., Hodges, S., Villar, N.: PreHeat: Controlling Home Heating Using Occupancy Prediction. In: Proceedings of UbiComp 2011. ACM (September 2011)
31. Seeeduino Film, http://www.seeedstudio.com/wiki/index.php?title=Seeeduino_Film
32. Shapeways, <http://shapeways.com>
33. Tinkercad, <http://tinkercad.com/>
34. TinkerKit, http://store.arduino.cc/eu/index.php?main_page=index&cPath=16
35. Tokuhisa, S., et al.: xTel: A Development Environment to Support Rapid Prototyping of Ubiquitous Content. In: TEI 2009, Cambridge, UK (February 2009)