

mbed



Rapid Prototyping for Microcontrollers

By Simon Ford, ARM

Microcontrollers are a solution looking for a problem, and that solution is getting impressive; lots of performance, lots of peripherals, and a price-point that opens them up to a host of new opportunities. However, the problem in exploiting these opportunities is exactly that; what is the problem?

Microcontrollers are getting smaller, more powerful, lower power, and more connected, yet the prices keep coming down. This is a huge opportunity for new markets that can successfully adopt the technology. Key to this is identifying the new problems that microcontrollers can solve, and building the proof-of-concept that carries an idea through to becoming a potential product.

The industry has built excellent tools for embedded engineers to productise microcontroller designs, once the desired specification is known. But when the task is to prove a concept or define a specification, even for experienced engineers the risks and timescales often don't add up. The result is that ideas don't get tried, there is little iteration or design space exploration, the final design is also the prototype, or the specification is overly cautious. Given that in many cases the ideas, observations and insights that could define these applications will be coming from people in other problem domains, these limitations are amplified. This has the potential to be a real barrier to adoption.

Strategy

The underlying goal of mbed is to enable efficient evaluation of microcontroller capabilities and prototyping of the applications they could be applied to. In particular, it aims to match the design cycle times of other aspects of product design. The strategy which helps realise this is to look for the technologies and trade-offs that can optimise the time to get to a working prototype, rather than optimizing the design itself.

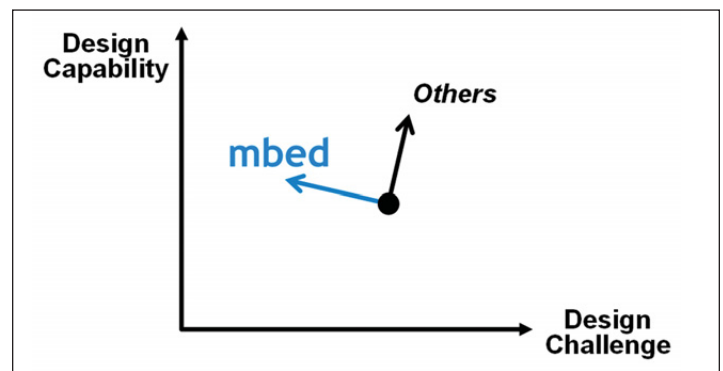


Figure 1: The mbed strategy

For example, the trend in microcontrollers is an increasing performance and memory capacity for a fixed price. Whilst most tools focus on enabling users to exploit this in the end capability of the applications that can be created, mbed instead focuses on using this performance and capacity to reduce the design challenge. A good example is providing high-level abstractions which make the functionality accessible at the cost of implementation efficiency and increased code size.

Another key objective is to overcome barriers to entry; the Technology Acceptance Model (Figure 2, next page) provides a clear framework to achieve this, highlighting ease of use and perceived usefulness.

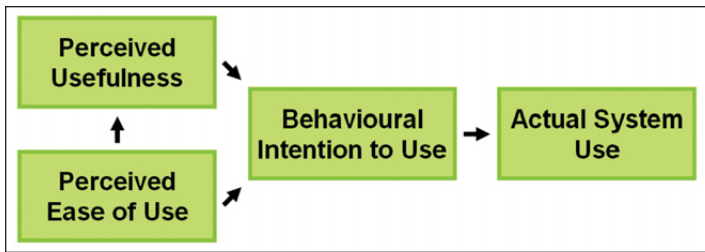


Figure 2: Technology Acceptance (Davis et al., 1989)

Perceived usefulness is driven by insight in to technology, through experimentation and education. The prototyping approach naturally supports user exploration, whilst enabling marketing and application engineering functions to easily demonstrate and showcase the technology.

Ease of use is actually very dependent on context; most good tools are easy to use for the task they were intended. But for a different task, the results are unlikely to be the same. By defining clearly the context of rapid prototyping, it becomes much more meaningful to make design trade-offs focused on ease of use. For an embedded developer in the industry familiar with an existing proprietary architecture and toolchain, a change can be daunting; with the additional negative feeling of going from expert to learner. These factors alone can be enough to put off exploring the benefits of moving to a modern solution. For a new user, the fear, uncertainty and doubt can be equally prohibitive. This makes the initial experience critical; the tools must give results quickly with little investment, building trust and earning any further ongoing investment.

Getting Started

A goal for mbed is to get a new user running their first program as soon as possible, to build confidence and trust in the hardware and software toolchain. The mbed tools have applied some novel technology to achieve this, and the results speak for themselves; you can get started in 60 seconds. This achievement means there is little excuse left not to experiment.

“The results speak for themselves; you can get started in 60 seconds.”

The results are achieved through two innovations; a USB disk based programmer on the hardware, and compiler tools as a cloud-computing based web-application that runs in a web browser. These solutions have some obvious benefits, but also some that aren't immediately apparent.

Up front is the advantage that there is nothing to setup or install. For many, this will simply come as a pleasant surprise, allowing instant access without the administrative duties. But for others with locked down computer systems, as found in many educational and workplace environments, this could be the difference between being able to test out a microcontroller or not.

The IDE is simple but functional, allowing it to step out of the way and do what it is designed for - edit and compile code. With everything pre-configured, it will work out-of-the-box, on any platform; PC, Mac or Linux. This instant no-hassle access gives great confidence in the tools, enabling users to pick them up whenever they need to use or demo them.

With it now common to work on a number of computers, the online approach becomes a particular advantage - not only do you avoid the problems of multiple installations and keeping them in sync, but your online workspace also comes with you wherever you are.

So an engineer or student may pick up at home what they started at work or university, just like is taken for granted with web-mail.

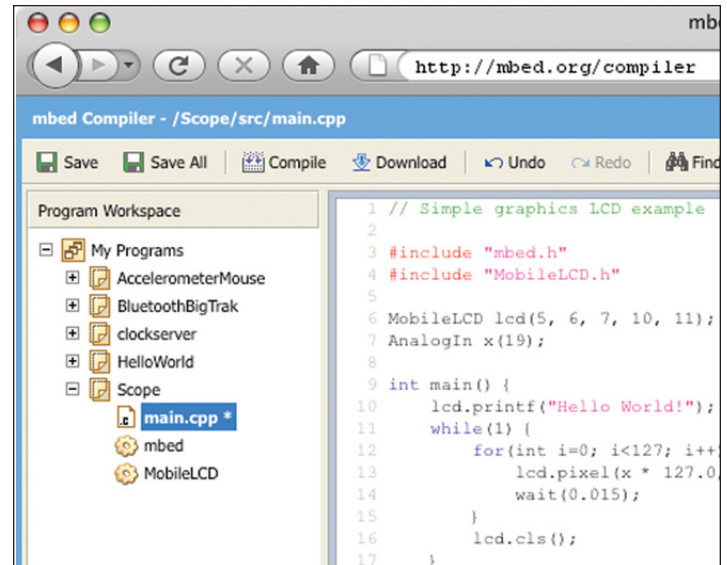


Figure 3: Online compiler

But the subtle advantages are the things you don't see. There are the decisions you don't have to make, because the options and configurations that will give the most appropriate results for the task have been made already.

Much of the hardware and software groundwork has been done. The tools are so lightweight that from any machine, it is possible to login, create a project from scratch and test out or modify something in a matter of minutes; that sort of flexibility can have a significant impact on working style. Having a simplified setup means everything is easily reproducible.

Combined with the single hardware and library model, every other mbed user is developing in an identical environment. That makes community support much easier, as people can share problems and questions with a common context.

“every other mbed user is developing in an identical environment...”

Rapid Prototyping

The architecture and implementation of the hardware and software components of mbed provide a unique advantage when it comes to prototyping.

The mbed Microcontroller hardware packages an NXP LPC1768 microcontroller, support components and smart USB interface in a practical 40-pin 0.1" pitch DIP form-factor, ideal for experimenting on solderless breadboard, stripboard and through-hole PCBs. To support the exposed interfaces, an mbed C/C++ library provides high-level interfaces to microcontroller peripherals, enabling a clean, compact, API-driven approach to coding. The combination gives immediate connectivity to peripherals and modules for prototyping and iteration of microcontroller-based system designs, providing developers with the freedom to be more innovative and more productive.

Figure 4 shows the basic mbed Microcontroller pinout, indicating the availability and location of the interface resources. The interfaces indicated match those found in the mbed Library. This highlights some of the key benefits of them being developed together. The API provides an abstract peripheral interface, rather than being implementation specific. The libraries use object-orientation which maps well to tangible physical hardware resources. The hardware, libraries and documentation share the same naming and concepts for interfaces.

The alignment between hardware and software enables a natural programming style that captures intent, essential for fast experiments and iteration.

For example, mbed avoids requiring the multiple levels of indirection that are usually needed for pinout and resource allocation; these tend to lose meaning and introduce bugs.

```
#include "mbed.h"

SPI myspi(p5, p6, p7); // mosi, miso, sclk

int main() {

    // Setup 9-bit SPI @ 1MHz
    myspi.frequency(1000000);
    myspi.format(9);

    int response = myspi.write(0x8F);
}
```

Figure 5: Configure and write to a SPI device

The SPI example in Figure 5 demonstrates setting up a SPI master interface. First, a SPI object is created and tied to the desired pins (mosi, miso and sclk), as chosen from Figure 4. Notice that this expression could now be equally useful when it came to physically wire up the device - the specification has captured the physical connectivity.

Next, the frequency and bit format of the SPI object (myspi) are configured, before performing a write/read transaction. The methods on the SPI object are well defined making the interface intuitive, and the operations are independent of the low level settings or requirements of the underlying hardware; in fact, to change the SPI port being used in this example, only the pin names would need to be changed. This helps separate modification of the physical

aspects of a design (the resources used and how they are pinned out) to the control (what they do).

A similar example of capturing intent is shown in Figure 6. In this case, a function is setup to be called every time a rising edge interrupt occurs on a digital input pin. Interrupts are a simple concept, but notoriously complex to setup and get functioning correctly. With mbed, the code is conceptually very simple; create a pin that can generate interrupts, and attach a function to the rising edge of that pin.

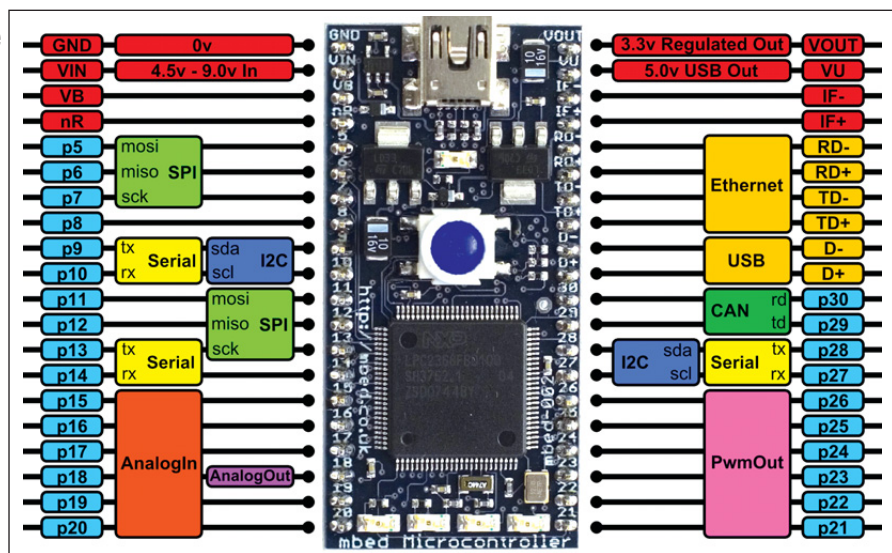


Figure 4: The mbed Microcontroller pinout

generate interrupts, and attach a function to the rising edge of that pin.

```
#include "mbed.h"

InterruptIn button(p5);
DigitalOut led(LED1);

void flip() {
    led = !led;
}

int main() {
    button.rise(&flip); // attach flip to p5 edge
    while(1);          // hang around forever
}
```

Figure 6: Attach a function to a pin interrupt event

A similar example of capturing intent is shown in Figure 6. In this case, a function is setup to be called every time a rising edge interrupt occurs on a digital input pin. Interrupts are a simple concept, but notoriously complex to setup and get functioning correctly.

With mbed, the code is conceptually very simple; create a pin that can generate interrupts, and attach a function to the rising edge of that pin. The library is built using these approaches throughout (see Figure 7), allowing developers to concentrate on application logic rather than implementation details.

Interface	Function
DigitalIn	Read the state of a digital input pin
DigitalOut	Write the state of a digital output pin
DigitalInOut	Read and write a bi-directional digital pin
InterruptIn	Trigger a function on a pin rising/falling edge
AnalogIn	Read the voltage on an analog input pin
AnalogOut	Control the voltage on an analog output pin
PwmOut	Control a pulse-width modulation output pin
Serial	Communicate with serial (UART) devices
SPI	Communicate with SPI slave devices
I2C	Communicate with I2C slave devices
CAN	Communicate on a CAN bus
Ethernet	Read and write Ethernet packets
Timer	A general purpose timer
Ticker	Call a function at a recurring interval
Timeout	Call a function at a point in the future

Figure 7: The mbed Library interfaces

The mbed Library is built on top of the low-level ARM® Cortex™ Microcontroller Software Interface Standard (CMSIS), which is a vendor-independent hardware abstraction layer for the Cortex-M processor series.

In contrast to CMSIS, the mbed Library provides a very high-level API, focused on providing abstract interfaces for basic control of peripherals. This structure provides a natural way for users to benefit from the mbed Library wherever they can, yet add bespoke code built on CMSIS where they need to support functionality not provided. In particular, this enables concentration of effort only on critical or differentiating aspects when prototyping. In addition to the mbed Library, the mbed Community peripheral libraries are an expanding base of contributed code for controlling peripherals that are connected to the microcontroller, such as sensors, actuators, LCDs and other modules. These are usually built on top of the mbed Library, and enable systems to be connected quickly, focusing on the logic and functionality rather than drivers. These libraries can be contributed by anyone within the mbed community, and will be supplemented by middleware from third-party vendors over time.

Application Example To demonstrate how a simple application experiment can be realized with mbed, the following example demonstrates a hardware device controlled by an internet database.

The example program in Figure 9 implements a system which displays a message on a screen and moves a servo motor based on the result of an HTTP request. The solution is unlikely to be optimal, robust or complete, but is enough to get the concept working.

The prototype may enable iteration of hardware, early development of the internet application, exploration of new markets or provide the case for committing to a project. By enabling an accessible way to test ideas, mbed helps reduce the risk associated with product development and gets advanced microcontrollers designed into applications more often.

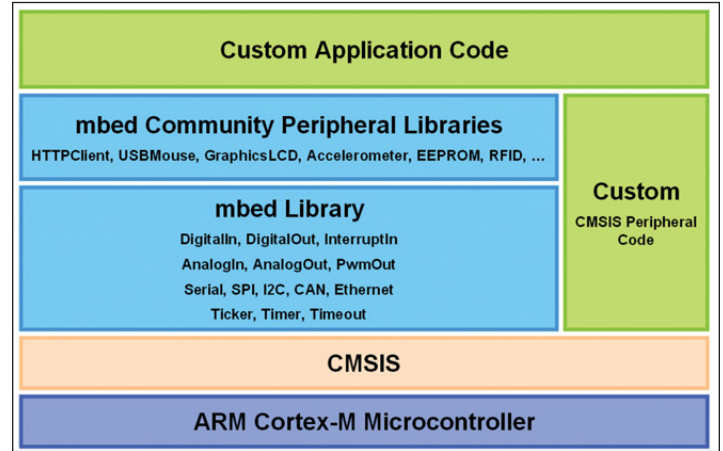


Figure 8: - mbed Library Architecture

```
#include "mbed.h"

#include "HTTPClient.h"
#include "MobileLCD.h"

MobileLCD lcd(p5, p7, p8, p9); // SPI LCD
HTTPClient http; // Ethernet client
PwmOut servo(p21); // R/C Servo

int main() {
    servo.period(0.020); // 20ms servo period
    char result[128];
    while(1) {
        http.get("http://a.com/stat.php", result);

        lcd.printf("The status is %s\n", result);

        // position the servo, 1-2ms pulsewidth
        float percent = atof(result);
        servo.pulsewidth(0.001 + 0.001 * percent);

        wait(60); // update every minute
    }
}
```

Conclusion The focus on rapid prototyping gives mbed a broad appeal. For engineers new to embedded applications, mbed will enable them to experiment and test product ideas for the first time. For experienced engineers, mbed provides a way to be more productive in the proof-of-concept stages of development. And for marketing, distributors and application engineers, mbed provides a consistent platform for demonstration, evaluation and support of microcontrollers. As a result, the mbed tools will help a diverse audience exploit the opportunities presented by advanced microcontrollers like the NXP LPC1768.

END