"Hands on" review by Tim Blythman

Introducing the chipKIT™ Lenny – a PIC-based Arduino

Arduino started out using 8-bit Atmel AVR micros, but these days, there are Arduinos based on all sorts of chips. This one happens to use basically the same device that Geoff Graham used in his 28-pin Micromite series. As you would expect, it's very capable, and it can take advantage of most of the vast range of Arduino software and hardware that's available.

The battle between Microchip and Atmel has been going on for a long time now, with neither side giving any ground; that is, until Microchip ended the argument by purchasing Atmel!

Despite that, to this day, we still see a clear line dividing the Atmel AVR-based Arduino boards and PIC-based boards such as the Micromite.

Even though Microchip took over Atmel in 2016, the two families remain essentially distinct, although some features have flowed between the two and you can now use Microchip's MPLABX IDE to program some Atmel microcontrollers.

The chipKIT family blurs this line further, allowing a PIC32-based microcontroller to be programmed with the Arduino IDE. The Lenny is only one member of this family; there are numerous other chipKIT boards with PIC32 microcontrollers and varying features.

They all sport a 32-bit PIC32 microcontroller, and with that comes all the advantages of a 32-bit microcontroller compared to the 8-bit AVRs. And like all PIC32 devices, they operate from a 3.3V supply, compared to the 5V that's typical for AVRs (although AVRs can run from 3.3V too).

To work with the Lenny, you'll need a copy of the Arduino IDE (integrated development environment), which can be downloaded for free from: siliconchip.com.au/link/aatq

chipKIT history

The first chipKIT boards were introduced around nine years ago by a partnership between Microchip Technology and Digilent. The idea was to create a PIC32-based board that could use Arduino-compatible add-ons (such as shields and modules) and also provide a programming experience for those familiar with the Arduino IDE.

The first boards were known as the chipKIT Uno32 and Max32, and were intended to be interchangeable with the Uno and Mega respectively. The Uno32 uses a PIC32MX320F128 while the Max32 sports a PIC32MX795F512, the same processor as the original Maximite (siliconchip.com.au/Series/30), also from around nine years ago.

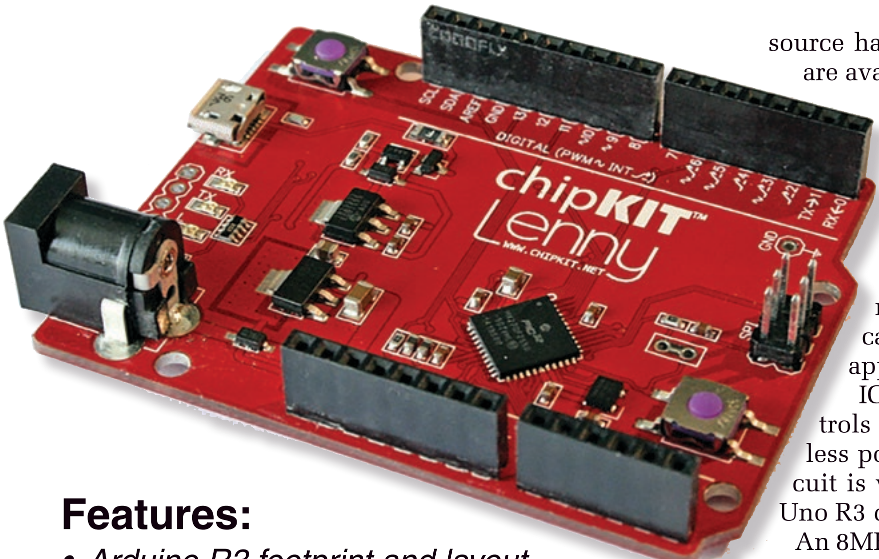
Several chipKIT boards have been developed, most using PIC32MX variants, although a few use the higher-specced PIC32MZ series.

Hardware compatibility is achieved by using the standard Arduino header layout, although there is the proviso that any attached boards must support 3.3V logic levels and not just 5V.

The PIC32 microcontrollers have a small edge here over some other 3.3V chips, in that many have some pins which are 5V tolerant, which simplifies interfacing to other 5V parts.

Much of the magic is in the software; the compiler and libraries mean that (for the most part), the same Arduino sketch code can be used for an ATmega328-based Uno and a PIC32 based chipKIT board.

The original IDE for working with chipKIT boards was called the MPIDE (multi-platform IDE) and was forked from the open-source Arduino IDE. Finally, with support for non-AVR boards being introduced into the Arduino IDE,



Features:

- *Arduino R3 footprint and layout*
- *32-bit PIC microcontroller (PIC32MX270F256)*
- *Native USB interface*
- *256kB flash memory (244kB usable)*
- *64kB RAM*
- *40MHz processor clock*

the chipKIT core for the Arduino was introduced.

So now, chipKIT support can be added to the Arduino IDE using the Boards Manager, after which the chipKIT boards appear in the usual list.

The chipKIT Lenny

While many of the early chipKIT boards were produced by Digilent, the open-source nature of the hardware and software meant that variants inevitably followed.

A company called Majenko Technologies designed the Lenny board; they specialise in open-source hardware designs.

We could have reviewed any of the chipKIT variants, but we chose the Lenny because it's one of the cheaper chipKIT boards available. It also appears to be well designed regarding Arduino compatibility. In particular, it follows the R3 layout. It has dedicated pins for I^{2}C and SPI in the correct places, as well as secondary I^{2}C and SPI connections where you would find them on the Uno.

So it has the best chance of working with shields, even if they date back to the days when the Uno was the only option.

It uses a PIC32MX270F256D micro. We used the DIP variant of this chip in our February 2019 USB Adaptor for Micros (siliconchip.com.au/Article/11414). Its immediate predecessor, the PIC32MX250F256B, was also used in the ASCII Video Terminal project from July 2014 (siliconchip.com.au/Article/7925).

These chips have an onboard USB peripheral. In this case, it is used for direct communication with the host PC, similarly to the Arduino Leonardo. And it's the Leonardo which is the inspiration for the Lenny design and name, in case you hadn't guessed.

The hardware

Fig.1 shows the schematic of the Lenny. As it is open-source hardware, all the design files (such as PCB files) are available online via siliconchip.com.au/link/aaxi

The DC jack, CON1, supplies up to 12V to 5V LD1117S50T regulator REG1 via schottky diode D1. Alternatively, REG1 can be supplied directly from the VIN pin. The 5V rail powers 3.3V MCP1825S-3302 regulator REG2.

The LD1117S50T regulator can handle up to 15V, but the Lenny manual notes an absolute maximum of 12V. Since 12V automotive systems can easily reach above 14V, this reduces the board's apparent usefulness.

IC2, an op amp configured as a comparator, controls Q1 to connect the 5V from the USB socket unless power is available from VIN. This part of the circuit is virtually identical to that used in the reference Uno R3 design.

An 8MHz clock is provided by clock oscillator XO1 and fed into the OSC1 pin of the PIC32MX270F256D, IC1. The pins of IC1 are broken out to the various headers around the board, as well as to the micro-USB socket, CON3.

There are two tactile push-buttons on the board. S1 is marked PROG and is used to activate the bootloader for uploading sketches, while S2 is used to reset the microcontroller.

Near the USB socket is the ICSP header (CON9) with staggered pins to allow a header to be friction-fitted temporarily. The ICSP header is not needed during regular operation, but can be used to program the PIC32 microcontroller directly or to update the bootloader firmware.

There are four LEDs onboard. Two indicate serial data activity (TX and RX), one is for power and one flashes during programming, and can be used for other tasks in your own code.

The usual array of bypass capacitors surround the microcontroller. While the board is sparse, the simplicity lends itself to the possibility of being the basis of other PIC32-based designs.

Table 1 shows the capabilities of each pin that's broken out to one of the usual Arduino headers.

Software

As mentioned above, to use the Lenny with the Arduino IDE, we need to install the chipKIT core. This contains several parts, but they are all installed as a single unit.

It includes a series of board definitions, which ensure that the pins marked on the board are correctly associated with the physical pins on each specific microcontroller.

It also includes a C++ compiler. Like the AVR Arduino core, it is based on the open-source gcc (GNU Compiler Collection). This turns the Arduino-flavoured C++ code into PIC32-specific machine code.

There are also libraries which translate the common Arduino-specific functions into code which correctly interfaces with the peripherals on a PIC32 microcontroller. This includes such simple functions as digitalWrite() and analogRead(), as well as things like the SPI and I^{2}C interfaces.

There are also utilities to upload the sketch to the board; in the Lenny's case, this is pic32prog, the same program that can be used to program some variants of the Micromite.

We're using version 1.8.5 of the Arduino IDE. To program the Lenny, you need an IDE new enough to include the Boards Manager, which was first included with version 1.6.7, but

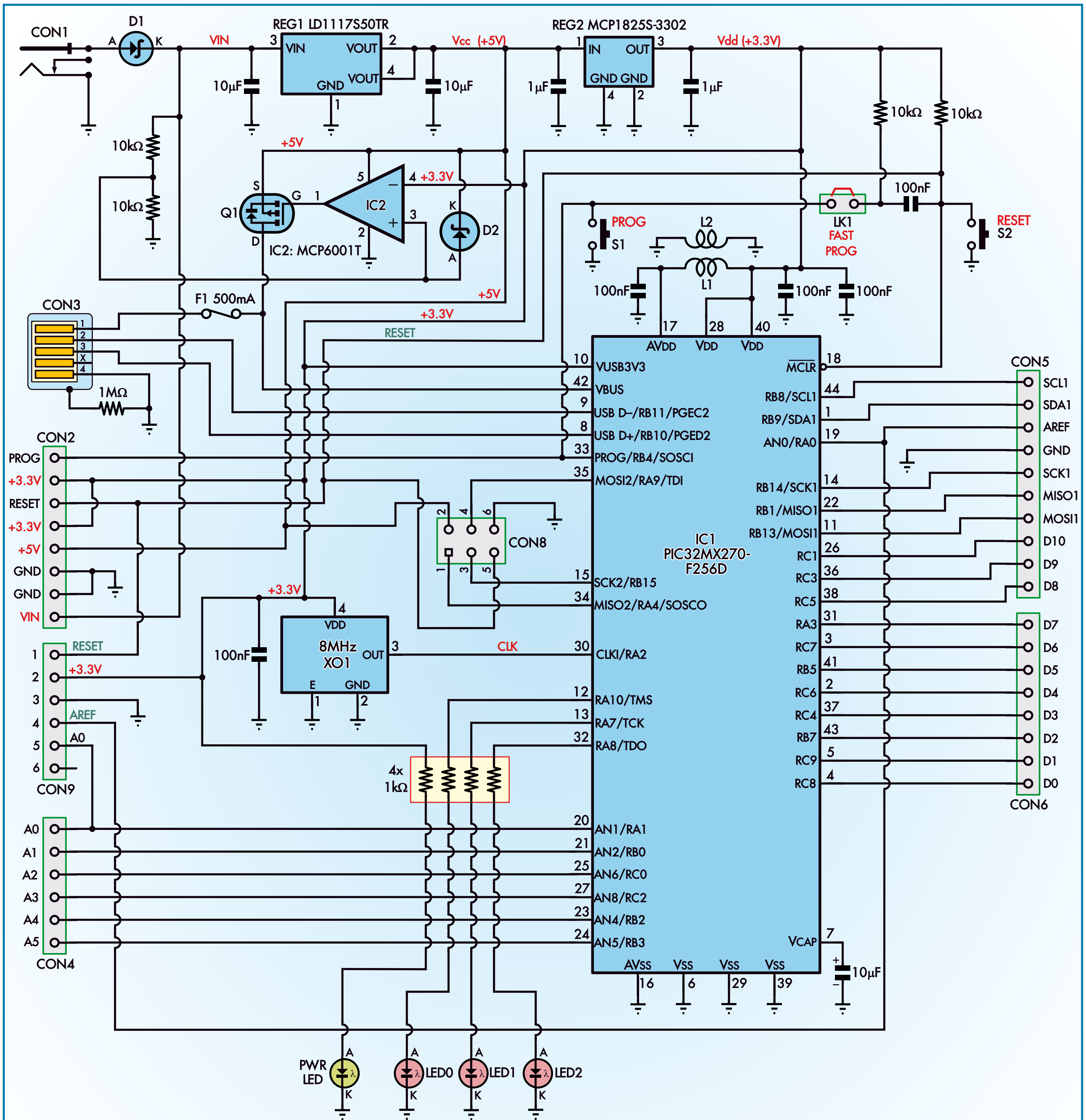


Fig.1: there isn't much to the Lenny circuit. The main microcontroller handles the USB interface, so there is no need for an interface IC. The remainder of the circuit is mostly involved with power supply and voltage regulation.

we haven't tried it with a version that early; it may work.

We're using Windows 10, but the same process should work for macOS, Linux (x86 and x64) and even a Raspberry Pi.

Board installation

Firstly, open the Preferences window (File -> Preferences) and add the following URL to the Additional Boards Manager URL list:

https://raw.githubusercontent.com/chipKIT32/chipKIT-core/master/package_chipkit_index.json

Separate this from any existing entries with a comma.

Now open the Boards Manager from Tools -> Board -> Boards Manager. It may take a few moments for the list to be populated, as each URL is checked. Unfortunately, you cannot remove URLs after the boards are installed, as this makes them unavailable from the IDE. We understand this behaviour may change in future versions of the Arduino IDE.

The chipKIT option should appear, as shown in Screen1, so click the Install button. Installation may take a while as

Pin	Features
D0	5V tolerant digital I/O, serial RX
D1	5V tolerant digital I/O, serial TX
D2	5V tolerant digital I/O, serial1 TX, interrupt
D3	5V tolerant digital I/O, PWM, interrupt
D4	5V tolerant digital I/O, serial1 RX, interrupt
D5	5V tolerant digital I/O, PWM, interrupt
D6	5V tolerant digital I/O, PWM, interrupt
D7	3.3V digital I/O
D8	5V tolerant digital I/O
D9	PWM
D10	PWM
D11	SPI MOSI
D12	SPI MISO
D13	SPI SCK
SDA	5V tolerant digital I/O
SCL	5V tolerant digital I/O
A0	analog or 3.3V digital I/O
A1	analog or 3.3V digital I/O
A2	analog or 3.3V digital I/O
A3	analog or 3.3V digital I/O
A4	analog, 3.3V digital I/O or I^{2}C SDA
A5	analog, 3.3V digital I/O or I^{2}C SCL
ICSP SCK	3.3V digital I/O
ICSP MOSI	5V tolerant I/O
ICSP MISO	3.3V digital I/O

Table 1 - Lenny pin capabilities

there are the various board definitions and compilation and upload tools to be installed. The total size is around 1GB.

Note that the Lenny board doesn't appear in the list on this screen, but it is supported by version 2.1.0 of the chipKIT core, as shown in Screen2.

Using it

We found that there are a few quirks when using the Lenny compared to a typical AVR board like the Uno. These are apart from obvious differences such as the 3.3V I/O voltage.

The first is the "PROG" button. The Lenny needs to be manually put into programming mode by pushing the PROG button, which isn't necessary on the Uno. This is because the onboard USB interface is shared between the programming interface and user programs. If you don't press this button before initiating a code upload, that upload will fail.

The PROG button can be a little awkward to access if a shield is fitted to the top of the Lenny. We were just able to get a finger into the gap, but we imagine some people might struggle with this. Also, note that this means that the serial port number (COMx) changes between programming mode and run mode.

This happens with the AVR-based Leonardo too, but the upload utility detects it, so it works seamlessly, and there is no need to change the serial port manually.

The Lenny software does not do this, so to work with a program that uses the serial port (especially for debugging), the serial port has to be changed twice for each program upload.

The way we sidestepped this is to use another serial terminal program, specifically, TeraTerm. TeraTerm has the advantage that it can resume communication even if a serial port disconnects while the terminal is open, as is the case when the Lenny switches to programming mode.

The TeraTerm window can simply stay open in the background. It operates a bit differently to the Arduino Serial Monitor, but it's perfectly adequate for most purposes.

Benchmarking

We decided to run some benchmarks on the Lenny, to compare its performance to other Arduino boards – see Table 2. We used the same method as in our review of the new Arduino Nano boards in October 2019 (siliconchip.com.au/Article/12015).

Since one of those boards, the Nano 33 IoT, also has a 32-bit chip (an Atmel SAMD21), this makes a good comparison for the Lenny.

The benchmark tests show the Lenny to be by far the fastest overall. Note that the Lenny runs at 40MHz while the Nano 33 IoT runs at 48MHz. The Nano 33 IoT is ahead by a tiny margin when doing byte, integer and long multiplies, but otherwise, the Lenny comes out firmly on top.

There are vague mentions of a 50MHz bootloader configuration for the Lenny, which we imagine would put it even further ahead. But the PIC on our Lenny is the 40MHz variant, so this upgrade is a bit dubious; it may work, but perhaps not reliably.

Compatibility

We don't expect that the Lenny will be immediately compatible with all Arduino sketches, in particular, those which use direct port writes. When such techniques are used, those sketches will only work on the specific board they are written for, which is typically the Uno.

To test this, we tried compiling a few different sketches written for different shields.

The first one we tried was for the Jaycar XC4454 LCD Shield. This uses the common HD44780-type LCD controller and is supported by the 'LiquidCrystal' library, which is usually included with the Arduino IDE.

Once we had the pins set correctly (the Jaycar shield uses a different pin configuration to the default), the sketch worked as expected. Since this shield uses one-way communication, it depends on the LCD controller accepting 3.3V logic levels, which it appears to do.

The next test was one of our own shields, the 3.5in Touchscreen Arduino Adapter from May 2019 (siliconchip.com.au/Article/11629). We found that the display worked fine, even with the level converting resistors in place.

The level-converting resistors are intended to allow 5V I/O signals to drive the 3.3V controller on the LCD, but in this case the 3.3V I/O is being divided down to 2.2V levels. So, it's remarkable that it worked!

The touch controller did not fare so well; we could not get it to work, even modifying the level-converting resistors to deliver 3.3V I/O signals (by removing the lower resistors from the dividers). We could not resolve this issue, but expect that there is some way to make it work. After all, the same display works perfectly well with the practically identical PIC32MX170F256 chip in the Micromite.

We suspect that this has to do with the different ways that SPI interfaces are handled, particularly as the touch

	Nano	Nano	EveryNano33 IoT	chipKIT Lenny
digitalRead	5.032µs	6.679µs	0.948µs	0.804µs
digitalWrite	4.532µs	6.459µs	1.913µs	1.066µs
pinMode	4.470µs	3.244µs	1.931µs	1.644µs
byte *	0.632µs	0.570µs	0.197µs	0.199µs
byte /	5.412µs	5.297µs	0.636µs	0.451µs
byte +	0.443µs	0.381µs	0.197µs	0.149µs
integer *	1.386µs	1.263µs	0.171µs	0.174µs
integer /	14.277µs	14.052µs	0.591µs	0.396µs
integer +	0.883µs	0.759µs	0.171µs	0.124µs
long *	6.102µs	5.547µs	0.168µs	0.174µs
long /	38.662µs	38.362µs	0.596µs	0.396µs
long +	1.763µs	1.514µs	0.169µs	0.124µs
float *	7.932µs	7.314µs	3.016µs	1.329µs
float /	80.162µs	78.337µs	11.721µs	4.296µs
float +	10.107µs	9.692µs	2.806µs	1.276µs
itoa()	12.957µs	12.792µs	3.041µs	0.876µs
ltoa()	125.987µs	125.487µs	16.196µs	2.696µs
dtostrf()	78.637µs	76.687µs		46.896µs
random()	91.412µs	90.512µs	9.546µs	2.121µs
yl=(1<<x)	0.569µs	0.444µs		0.099µs
bitSet()	0.569µs	0.444µs	0.123µs	0.099µs
analogRead()	111.987µs	112.887µs	422.946µs	21.046µs
analogWrite()	7.167µs	6.932µs	6.801µs	1.401µs

Table 2 - chipKIT Lenny benchmark (lower is better)

controller IC on the 3.5in panels works at a much lower maximum bus speed than the LCD.

We also tried our updated Seismograph shield from April 2019 (siliconchip.com.au/Article/11532). Amongst the other hardware, the main shield used has an SD card interface that uses the SPI peripheral and a real-time clock (RTC) module that uses the I^{2}C peripheral.

This project did not compile immediately, as we used a specific format of an I^{2}C command that had not been implemented in the chipKIT core.

This format (where a third argument is presented in the requestFrom() function call) is documented in the official Arduino reference. Further investigation shows that this issue has been identified but not fixed in the chipKIT core (see https://github.com/chipKIT32/chipKIT-core/issues/240).

The specific SD card file system library we used in this project was not able to read the SD card either. We also tried an example SD card sketch (CardInfo) from the Arduino IDE, and this was able to correctly identify the card and list its contents.

So it appears there are some minor differences between the AVR and chipKIT libraries.

PWM support

You might have also noticed from Table 1 that the Lenny only has five hardware PWM pins, compared to the Uno's six; pin D11 is the one that is missing this feature. Fortunately, the extra speed of the PIC32 microcontroller means that software-based PWM is available and can perform this task instead.

The 'SoftPWMServo' library uses the core timer to generate PWM signals (and servo signals) on pins that do not have hardware PWM support. The library notes that it may be subject to approximately 50ns of jitter in the output.

This equates to around 1% of the pulse width resolution, so is unlikely to be noticeable for most applications.

Special features

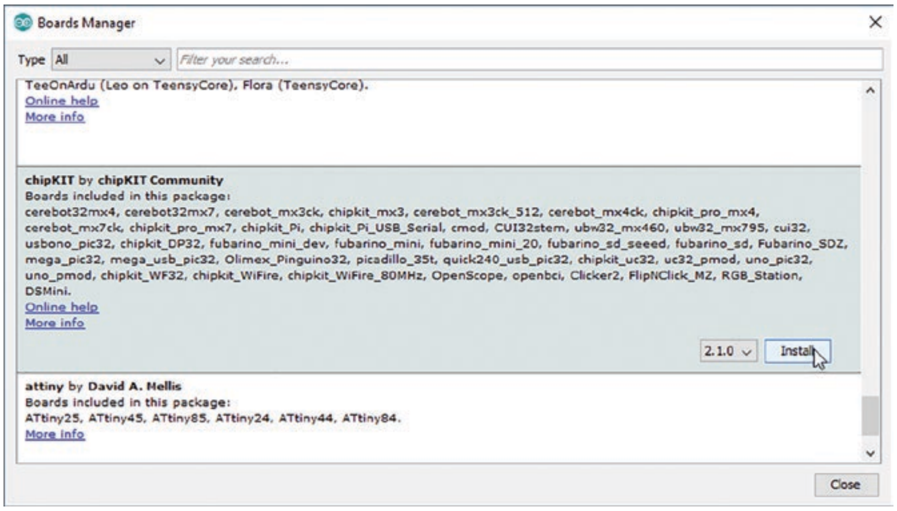
While browsing through the list of included examples, we noticed a folder called "USB_MSD". Inside, there are two example sketches which program the Lenny board to behave as though it is a USB Mass Storage Device.

At first, we could not get either of the examples to compile, but by adding two lines (and commenting a third out), we got the sketch "AnalogToFile" to compile and upload. These changes are shown in Screen3.

Once uploaded, the Lenny was visible to the attached computer as a USB Mass Storage Device. After formatting it, we were able to copy files to it. There was only 26kB of space available, as the contents are held in a 48kB RAM buffer. The "AnalogToFile" sketch also creates a file in this file system, which can be read by the USB host computer.

Being able to program a board to emulate a USB stick that can modify its own contents is very interesting. Previously, to copy log files from an Arduino project, you needed an SD card or a clunky custom interface, such as copying data from a serial terminal.

Now, using the Lenny, you can simply get the board to

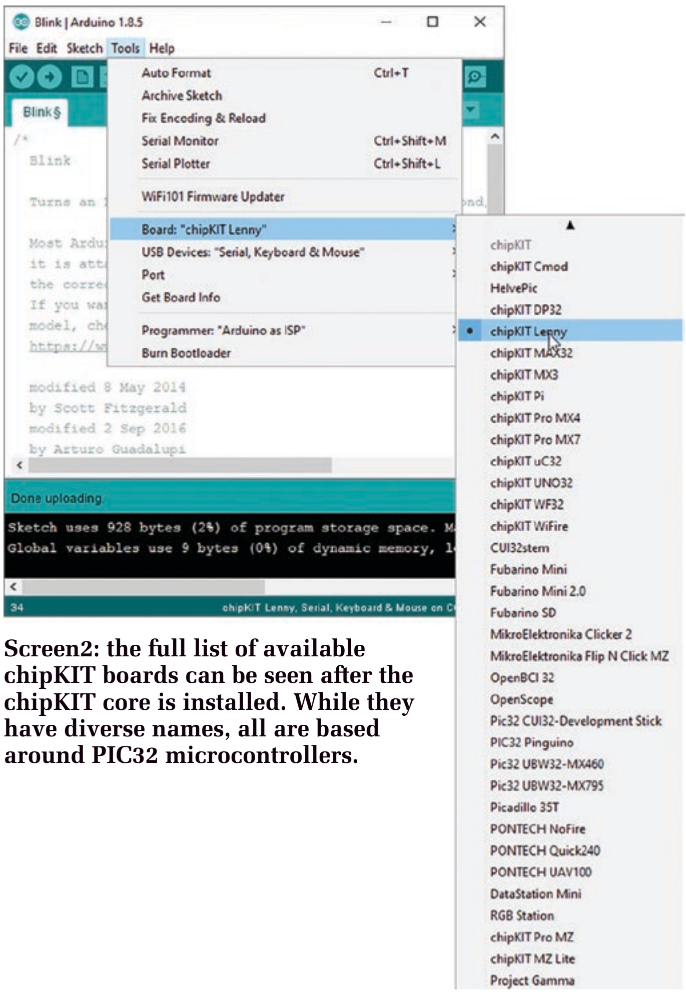


TeeOnArdu (Leo on TeensyCore), Flora (TeensyCore).

chipKIT by chipKIT Community
Boards included in this package:
cerebot32mx4, cerebot32mx7, cerebot_mx3ck, chipkit_mx3, cerebot_mx3ck_512, cerebot_mx4ck, chipkit_pro_mx4, cerebot_mx7ck, chipkit_pro_mx7, chipkit_Pi, chipkit_Pi_USB_Serial, cmod, CUI32stem, ubw32_mx460, ubw32_mx795, cui32, usbono_pic32, chipkit_DP32, fubarino_mini_dev, fubarino_mini, fubarino_mini_20, fubarino_sd_seeed, fubarino_sd, Fubarino_SDZ, mega_pic32, mega_usb_pic32, Olimex_Pinguino32, picadillo_35t, quick240_usb_pic32, chipkit_uc32, uc32_pmod, uno_pic32, uno_pmod, chipkit_WF32, chipkit_WiFire, chipkit_WiFire_80MHz, OpenScope, openbci, Clicker2, FlipNClick_MZ, RGB_Station, DSMini.

attiny by David A. Mellis
Boards included in this package:
ATtiny25, ATtiny45, ATtiny85, ATtiny24, ATtiny44, ATtiny84.

Screen1: once the chipKIT URL has been added to the preferences page, the chipKIT core can be selected from the Boards Manager. Though the Lenny is not in this list, its profile is installed.



```
/*
  Blink

  Turns an

  Most Ardu
  it is att
  the corre
  If you wa
  model, ch
  https://w

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
```

```
Sketch uses 928 bytes (2%) of program storage space. M
Global variables use 9 bytes (0%) of dynamic memory, l
```

Screen2: the full list of available chipKIT boards can be seen after the chipKIT core is installed. While they have diverse names, all are based around PIC32 microcontrollers.

generate its log file to the internal RAM image and then it can be easily copied and pasted via a file browser program.

We haven't looked into this too deeply, but there is probably a way to attach to an SD card and just use the Lenny as a card reader that can also write to itself. While it is a limited and simple interface, we think there are many potential uses for it.

Other USB features

Some other USB-equipped boards like the Leonardo can emulate a keyboard or mouse. We used such a board (called a 'Beetle') in our project from August 2018 to interface an IR remote control to a computer by emulating a keyboard (siliconchip.com.au/Article/11195).

We tried the keyboard and mouse examples that are available for the Lenny, and they performed as expected, although we could not get an infrared interface working, as the IR library uses AVR-specific interrupt code to receive the signal.

The overall impression here is that most things will work, but you need to set aside some time to work through the minor niggles which pop up. The chipKIT core files are common to all chipKIT boards; thus, it would be a similar process to get such shields working with any chipKIT board. But unfortunately, some libraries depend on AVR-specific features, so they cannot be made to work easily.

The Lenny verdict

We're impressed with the speed of the Lenny board, as shown in the benchmarks. The 32-bit processor is much faster at mathematically-intensive programs than an 8-bit processor, and generally quicker than other 32-bit boards such as the Nano 33 IoT.

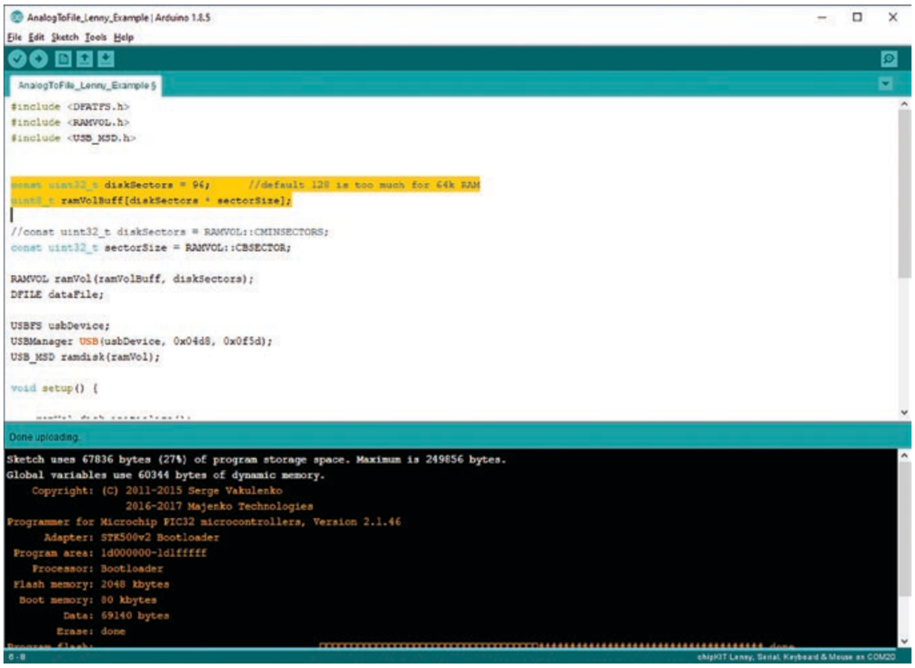
The extra speed also means that software PWM on all pins is possible. While not as accurate as hardware PWM, it is certainly adequate for most purposes.

The writers of the chipKIT core have worked hard to make it compatible with other Arduinos, but there are still some gaps present in important libraries. So it is not always a trivial process to port existing projects from 8-bit AVR-based boards to the Lenny.

The ability of the Lenny to behave as a USB Mass Storage Device is really powerful, since it is such an intuitive way to move files around.

Overall, the Lenny is a great board, but perhaps not capable of being a drop-in substitute for AVR-based boards. We expect that it will be best used in applications where its specific features would be a benefit over other boards, rather than as an upgrade in existing applications.

In particular, we expect to see projects spring up around USB Mass Storage Device examples. SC


AnalogToFile_Lenny_Example | Arduino 1.8.5
File Edit Sketch Tools Help
AnalogToFile_Lenny_Example §
Done uploading.
6-8
chipKIT Lenny, Serial, Keyboard & Mouse on COM20

```
#include <DFATFS.h>
#include <RAMVOL.h>
#include <USB_MSD.h>


const uint32_t diskSectors = 96;      //default 128 is too much for 64k RAM
uint8_t ramVolBuff[diskSectors * sectorSize];

//const uint32_t diskSectors = RAMVOL::CMINSECTORS;
const uint32_t sectorSize = RAMVOL::CBSECTOR;

RAMVOL ramVol(ramVolBuff, diskSectors);
DFILE dataFile;

USBFS usbDevice;
USBManager USB(usbDevice, 0x04d8, 0x0f5d);
USB_MSD ramdisk(ramVol);

void setup() {
```

```
Sketch uses 67836 bytes (27%) of program storage space. Maximum is 249856 bytes.
Global variables use 60344 bytes of dynamic memory.
    Copyright: (C) 2011-2015 Serge Vakulenko
               2016-2017 Majenko Technologies
Programmer for Microchip PIC32 microcontrollers, Version 2.1.46
      Adapter: STK500v2 Bootloader
 Program area: 1d000000-1dlfffff
    Processor: Bootloader
 Flash memory: 2048 kbytes
  Boot memory: 80 kbytes
         Data: 69140 bytes
        Erase: done
```

Screen3: the "USB_MSD" examples show off what we think is one of the Lenny's most interesting feature, being able to act as a USB Mass Storage Device. We needed to make some minor changes to the code to get it to compile, which are shown here.