Building Computer Systems for the Internet of Things

using the Arduino, ARM, x86 and MSP430 chipsets

by Evan J. Williams

New Holland Press
Princeton, 2019

ISBN-13 978-0-578-43802-3
21 Lincoln Avenue
Princeton, NJ 08540
908-359-8070
editor@newhollandpress.com
www.newhollandpress.com

Dedicated to,

The Makers and Menders of
The Internet of Things

Preface

The Internet of Things is an amazing development in the history of the Internet. It is the ability of ordinary devices such as wrist watches and water pumps to report, be accessed and controlled through the Internet. In this book I shall paint a spectrum of the development of the Internet of Things and explain how the individual developer can benefit from this trend.

Simply put, the Internet of Things (IoT) is a network of connected computer that control or monitor real-world actvities. IoT is more than networked computers: specifically it is all about specialized, dedicated or "embedded" processors managing a certain task. The things of the Internet of Things can be anything, like a thermometer or a light switch, but are run by computers that can send data to a decision-maker or respond to requests. The new functionality this adds to devices makes the Internet of Things into a networking tool capable of finer control of critical applications than stand-alone machines.

The Internet of Things is a big topic. Just as twenty years ago the computer hobbyist might seek out new space on the Internet, today the Do-It-Yourselfer might construct or use their own devices in the Internet of Things landscape. You might say that we just need to know what IoT is in order to benefit from it, and not need to go deeper with our understanding or actually build new devices. But to reuse a common analogy, we drive a car without rebuilding the whole thing ourselves. We don't have to know how the car operates to drive it. Looking at the components of IoT makes us a little more

knowledgeable about what is going on. Programming the computer helps us to use it.

Since this book originated as a talk given at The College of New Jersey for the Trenton Computer Festival in 2018 it makes sense that I take a conversational or story-like approach to this topic. The reader in applied technology wants to hear stories in this way from people like me who have enjoyed working with computers.

I have been able to organize the several chapters of this book into sections that reflect the stages of development of small computers that I have worked on, over time, throughout the years. That is why the chapters range from specific computer projects to generalities about the operating systems in a book that titled "Building Computer Systems for the Internet of Things" would appear to be about computers alone. Firmware? When there have been subjects such as Field Programmable Gate Arrays (FPGA) that are new to me or very technical I have simply provided an explanation of these features or devices.

My goal is to give you the tools to get up and running with IoT. For this book is intended to be a workbook for myself and others who are creating electronic things and completing whole, functioning units. By providing a variety of subjects and disciplines into which one can dive more deeply if need be,. I liken my role to that of a System Integrator who is creating a computer system and considering all the computers and software that will be used, from the cellphone to the server.

I start out in the Introduction with a look at the Internet of Things. I explain what it is and why it is different today from yesterday's Computer technology. Then in Chapter Two, I discuss some of the tools used to design and build Internet of Things projects. Then we walk through several implementations of the Blink program in Chapter Three, moving on to discuss processors for the Internet of Things in Chapter Four. In Chapter Five we build an actual device that is Internet of Things ready, an Arduino Thermometer. Then we discuss more devices, communcation protocols and Bluetooth in Chapters Five through Seven. Chapter Eight is the cumulation of this effort as we are then able to build an entry-level IoT example (Blynk)

using the Arduino Uno. Why have we chosen to use a small board like the Arduino rather than a larger board like Raspberry Pi or one of the Intel IoT computers that supports an IoT operating system, we ask ourselves in Chapter Nine? Our next examples, concept sketches for an IoT Inventory Control System and an IoT Ultimate Frisbee in Chapters Ten and Eleven make it more clear to us than before that our design goals may have a large influence on the processor we choose. We saw this in Chapter Five already when we asked whether we needed to use a processor at all for just a thermometer.

Finally we round out this small volume with some of the other tools that can help us on our way. This includes the Operating System and Software Tools for kernel development, which, believe it or not, we can undertake ourselves. The Internet of Things Dashboard, IoT Security, Power Requirments and 3D Printing are also concerns of ours as we reach the conclusion.

In this book we meet the real building blocks of Internet of Things devices as they exist: the processors, the displays, the components and circuits, the sensors and the communications devices. In Chapter Four we delve into processors and the commonly available small board computers built from them. Central to the operation of an IoT device is the processor. We go on to start projects using these new tools. So far this is all hardware, but at this point we cannot leave the software we are using out of our discussion and we review communications protocols and the operating system itself.

It is fitting to ask how we might program our devices and this question also takes several turns for we might have a choice between building our own operating system (for an UDOO x86 system) or writing some embedded C++ code (for Arduino development). I finish the book with a variety of chapters and appendices on some helpful tools.

I have implemented systems using many of these toolkits. It is my intention to write a book that will help others as other writers have helped me, benefitting from wisdom and 'know-how' as much as academic excellence.

I must thank three people who helped me to make this book a reali-

ty: Ms. MaryLouise Alu Curto, for her insightful comments, Dr. Allen Katz of The College of New Jersey, who included me in the Trenton Computer Festival for now the third year, and my mother, Ms. Idaherma Williams, . I have dedicated this book to the Makers and Menders of The Internet of Things.

Evan Jan Williams
Princeton, NJ
January, 2019

Table of Contents

1

Introduction

Perhaps we have been missing out on the new revolution in technology. We are starting to speak rather glibly about the new "Internet of Things" in the same way that we spoke of the World Wide Web 25 years ago and "Web 2.0" not so very long ago . Meanwhile, inventors and manufacturers of all types are learning of the ability of 3D printing to recreate our world. The "Things" that we need for the Internet of Things (IoT) we can practically create on the other side of our basement using additive machining technology. And why is the production of things and equipment, and the people who create them, as important as the things of the Internet of Things themselves? Our culture and our new awareness to the environment demands that we be in touch not just with people but also the things of our lives and that of others. We must identify what the Internet of Things is and how it is essential to our progress as a culture now.

What is a Thing? We can say a thing is "an inanimate object distinguished from a living being." But in a surreal way we are starting to talk about "smart things," that is coffee pots and refrigerators and thermos that are said to "know" when it is time to "brew," "reorder supplies" or call your cellular phone. Like advertising from the 1950s or the initial promises of the Internet in 2000, these improvements to ordinary things are supposed to be revolutionary in concept. But at their essence the Internet of Things are ordinary real-world devices whose data can be accessed via the Internet and other networks.

On their own, Smart Things are just as important as the Internet of Things. But just as "no man is an island," nothing can exist on its

own for very long and still be considered smart. Yet we include in our survey of the Internet of Things, smart tools and devices that are not necessarily always connecting back to the Internet for more information. One hallmark of Smart Things is that they can use machine learning algorithms to make some decisions for themselves.

This network of things is important because in a world of smaller industry they allow devices and people to make more informed decisions about their actions, such as reordering supplies or lowering the temperature, within their ecosystem. These things allow us to handle a new world in which technology is no longer central to our lives but is guided more readily by makers and menders such as you and I.

Without being overly critical of our human nature to hold high and sometimes unrealistic expectations, the Internet of Things revolution occurring today is critically important to our ability to maintain our standard-of-living and continue to progress as a civilization. The Industrial Revolution has long past, but we still need mechanical and electronic devices. This book will show you how to choose the basic building blocks to build your own devices for the Internet of Things.

What is the Internet of Things?

What is the Internet of Things? It seems to be the latest fad. At first glance one might properly answer that the Internet of Things are ordinary real-world devices that are enabled to be accessed over the Internet. If you gave that answer you would be right! But the real importance of the Internet of Things is to allow connected devices – and people – to make better decisions within their environment.

We all know what the Internet is. A global public network of connected computers. But what are the Things of the Internet of Things?

My cell phone may be a 'thing' but I am not a thing. I may have so many gadgets that the Internet may think I am another computer! A thing is not just anything. A thing for the Internet of Things can only be a thing if someone can inspect its status on the Internet. This is merely logic: not a rule of any sort. These tautologies may seem to

be trite but just start to build a database of things; then you will find out! A picture frame, even a digital picture frame, is just a thing unless it is controlled by a person or program on the Internet. A video camera with an Internet address is a thing of the Internet of Things.

We must realize that we do not think like smart machines but we still may be counted or outsmarted by them. As with all new technology, however great, we should be conscious of the insidious side effects.
The term Internet of Things is just a convention. There are already a lot of ways in which communications occur over the Internet between devices and people, such as a remote weather station that can call a cell phone, for example. But while you are considering what sort of things might be managed by Internet of Things devices perhaps you would also do well to consider how many things are projected to be in the Internet of Things universe.

A not-so-recent report said that there might be five billion Things by 2020. But before we even question the meaning of this statistic, let's also think about the current Internet address space assignments either in IPV4 or IPV6 and the impact that the resources used by only one Internet of Things thing might require. A lot of cloud computing power is required to handle these devices.

Then too, it is important to consider that the Internet of Things is also tied in with its developer communities, which include Makers and Menders, people like you and I who today have the biggest impact on how smart things are created and used.

Makers and Menders

There is a certain satisfaction to building things yourself. Building something yourself may be the best way to obtain anything from radios to computers to furniture. Our culture is more interested in recycling, repurposing and reuse than ever before. What used to be a throwaway society is now fashioning new items out of old things. It is harder than before to go to a store or hobby shop and obtain the small parts you need for projects.

Saturday March 11th, 2018 was New Jersey Maker's Day, a statewide

event in New Jersey, USA with a website at http://www.njmakersday.org. Meanwhile a larger community headed up by the Maker Faire organization also sponsors events in the United States.

Makers are building things themselves that aren't readily available in stores or are too costly. With advances in 3D printing we are now more likely to be able to print and build what we need within our community than order it on the web. Community based printers such as 3D Hubs at https://www.3dhubs.com/ are giving ordinary people the chance to both print 3D objects or provide printing services to others. And smart things are capable of telling other computers when it is time to reorder. It's not to far-fetched now to imagine a world run by robots who reorder and rebuild their components by themselves.

Perhaps the great promise of the Internet as an equal exchange between ordinary people and not just big companies are being fulfilled by the Maker movement.

Menders

You may be asking "Who is going to fix all of these things made by Makers?" In that case you are probably a Mender and not a Maker. There is no nationwide or statewide Menders day, unfortunately. Hmm, sound familiar? Maker or Mender, you might find more value in your old stuff than you thought was there.

There is a lot of talk of Makers. These are the people who are building new things for the Internet of Things. Not so many people are speaking of Menders; the people who fix the things that are broken or maintain existing equipment. Everyone wants to move to the "top-of-the-class" so while you won't be finding too many new projects built on Visual Basic 6 right now, for example, you may find a lot of people patching up old VB6 code.

Some companies (Corgibytes www.corgibytes.com) makes It their business to mend and heal software. They are more focused on repairing and rebuilding existing software than starting new projects. This is a company that specializes in fixing computer programming projects that have gone wrong. But there are a lot more areas in

which Menders are needed.

An article in The Economist bewails the fact that no one repairs things any more. (https://www.economist.com/blogs/schumpeter/2014/06/repair-caf-s). But just as the paradigm of Makers is new to us, and not quite the same as the Do-It-Yourself movements of the past; Menders are not necessarily Fix-It people but Engineers who redesign and refactor existing products.

A Gender Bender

What kind of headers do you feel are better on a computer board? Male or Female? Do you know the difference? Let's talk about flowers for a minute. Flowers have different parts, male and female, and depending upon the plant how the flowers enable the plant to reproduce is different. The same goes for computer parts. Sooner or later a part ends and another one begins. What do you do? You can either solder wires together or solder on a plug and plug the plug in somewhere! But don't laugh! This stuff matters.

Now suppose that you have twenty wires from a component or sub-circuit board that must be plugged into a computer processing board. What is better? Male or Female Headers. The answer is obviously male headers. Why? Because most ribbon cables come with female headers already crimped on the end.

Arduino UNOs and other Arduino's and most of the Maker boards (except Texas Instruments' MSP430 --https://www.youtube.com/watch?v=ZUYM25UDOg0) have Female headers. These are better for the experimenter because they don't have worry about shorting leads together or having a fancy connection wire when they want to try an experiment!

But wait! Are there any other reasons? Texas Instruments' engineers were able to work a lot better with male headers. The existing female cable from Logic Analyzers could be plugged into male headers. Also, oscilloscopes and other probes can grab hold of the male headers a lot better. There are a lot more reasons to debate this topic.

I think that having female headers on board the small computer

boards like Arduino UNO are preferable because not only can experimenters plug ordinary wires into the female leads but there are no exposed pins on these expensive boards that can bend or break as easily as the male pins can.

But wait! Have you looked at what Texas Instruments' MSP430 series can do?! These engineers know what they are talking about when it comes to core development of circuits and boards. So, before you rush off and buy the ESP8266, Arduino or Raspberry Pi, check out TI!

Bare Metal and Other Expressions

We might all be headed back to the kitchen table or garage to start building new things. The computer industry wasn't always dominated by the male gender. Lady Ada, a mathematician of the 1800s is credited with creating the first computer program. And women ran the divisions in the Army that created computer technology . But forty or so years of computer geeks may have overturned the playing field. Now we have to deal with phrases like "Bare Metal" and "If it compiles and links it runs," if we are to understand each other. Imagine then how hard it is to read computer programs written by these people! The back-to-basics movement of the Internet of Things, and new programming languages like Circuit Python and Kotlin give us a chance to correct some of these poor choices in direction. These changes in the direction of technology are more important than today's gender wars in the board room. Perhaps we are making better choices because of increasing gender equality in technology.

A Short History of the Internet

Most discussions of technology eventually include a 'history' that is retold to explain how we got into the place we now find ourselves. An example of such a history is Christopher Barnatt's excellent book. These histories are created for people and are invariably part of an analysis such as this. But they have nothing to do with how the computer works! Remember, that the things of the Internet of Things have been around a while; now they are Internet enabled and they are smart enough to interact with the human world. But these things are still working devices that operate whether a history is included

in the README or not!
We might go back to the first days of 'distributed computing' to discuss the changes that have made the Internet of Things a possibility today. After mainframe computers became available, people and companies still needed to obtain the results of calculations. Whether or not they were used to provide sales forecasts, these early computers printed their results on punched tape and reams of printouts and sent by courier across campus to their clients. As far-fetched as this notion seems today, the more recent days of 3-tier client server architecture are not far behind us.

In traditional 3-tier server architecture a computer at the 'back-end' does the bookkeeping required to store data and accounts that clients, perhaps Windows machines send over from the 'front-end.' Although this used to be an interaction that took place between dedicated software systems, now to a large degree these 'transactions' occur on the front-end in the browser and on the back-end at the Internet server.
In the Internet of Things there still is a server (aka Internet Cloud), but the browser has been replaced by a variety of devices such as a cellphone and not necessarily a Windows workstation or Apple Macintosh.

Design First!

If I have learned one thing over the years I have experimented with computing devices and built projects, it is to design first! One can't always do that however. Sooner or later one needs to take the parts one has in stock and put them together. But once the way is clear it is good to create a design and complete it.

What Have We Learned as People?

Don't forget we are the people served by the Internet of Things. We are the people who build and develop and use these systems. How are we learning to cope, as people, with a new world of Smart computer-enabled 'Things'? These 'Things' are making suggestions to us such as what beer to drink. Are we still able to discern our own way?

Upcoming Chapters

In the chapters that follow we will outline how the pieces of the Internet of Things become part of fully working systems. In Chapter Two we introduce some of the software tools used for the Internet of Things.

2 What is an IDE?

An Interactive Development Environment (IDE) is essential to most software development work. Although one can use the simplest of text editors such as Notepad in which to write programs, an IDE gives one to manage multiple files and even compile and test the final program. Depending upon the application or embedded processor one uses different IDEs.

The Arduino IDE, available from arduino.cc is an example of a simple interactive development environment with a bevy of useful features. It can be used to write and execute the simplest "sketches" that are used by the Arduino Uno and other boards. But a lot goes on behind the scenes when one clicks compile and run. The Arduino IDE can be used to load different libraries and program vastly different boards that support the Arduino standard.

Eclipse-based IDEs, from eclipse.org are the most popular IDEs around. Open-source by design they are used by a lot of vendors as a basis for their own IDEs. The Beryllium platform for Arduino is an example of a project that provides an alternative means of compiling Arduino code.

With enough skill one can build one's own cross-compilers for Arduino, ARM and TI MSP430 chipsets. But it might be a lot easier to use TI's free version of Eclipse for programming their boards or a proprietary ARM IDE. Atmel Inc., who makes Arduino UNOs and other Arduinos has their own IDE whcih I would like to try but as of now I am out-of-space on my desktop.

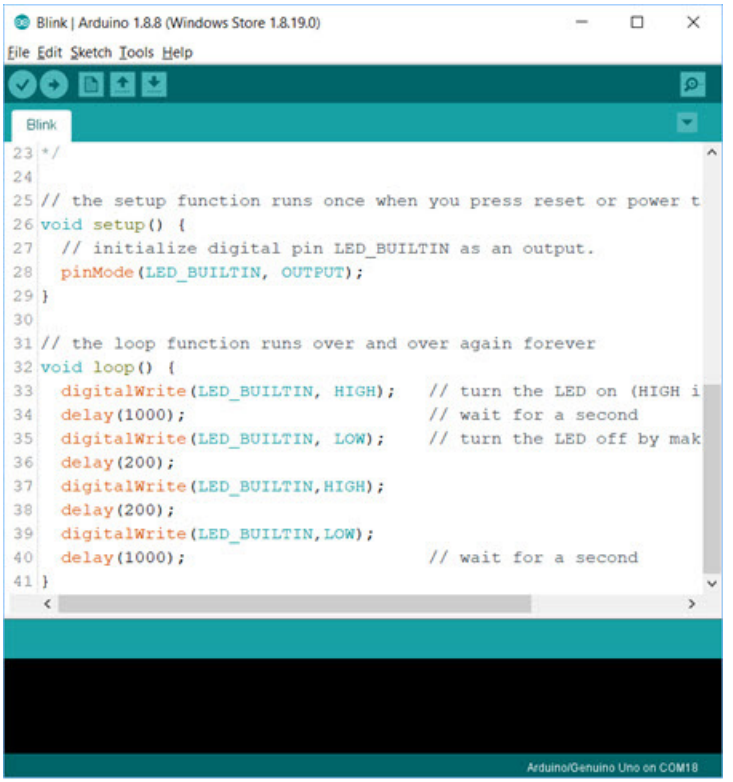
Speaking of the host environment, one needs a fairly robust system to run these compilers and toolkits. Microsoft Windows is the hands-down winner for operating such programs unless one has

the skill to run the IDEs on a linux system. Even then one must be careful with the version compatibility of Windows and the IDEs one wants to run. Although it is a best practice to always used the latest version of Windows, a lot of engineers run a revision back in order to keep their essential programs running. Even the new motherboard hardware such as USB ports may have some backwards compatibility issues.

Installing an Integrated Development Environment

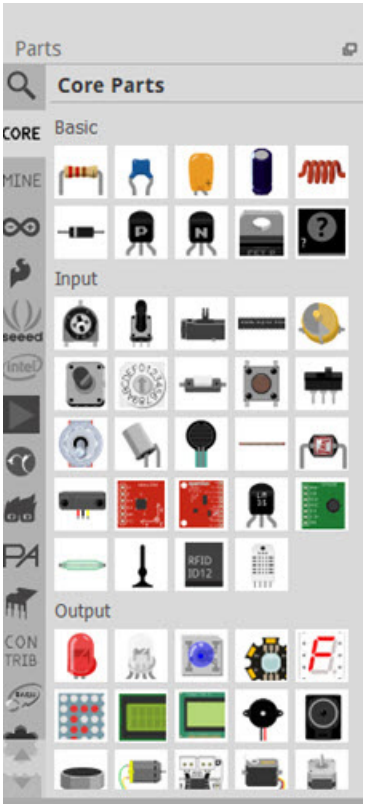
It is pretty easy to tell someone that all they need to do is download the Arduino IDE from arduino.cc and install it, but if you have never done this before it may be complicated. Rather than going through all the steps for a sample install in these pages I would rather mention that a lot of software packages come with installation instructions or other online help. There are so many diffferent kinds of laptops and desktops that it is difficult to always have the right prerequites for software installation. Good software however does often contain a list of these requirements so one can possibly avoid going too far down a path that is not likely to be profitable, like installing the Arduino IDE on Windows Vista, for example.

Of course, the purist will skip the IDE's altogether, build their own AVR cross compilers and upload the resulting binary files directly to the UNO using 'avrdude.'



Circuit Layout Programs

Working with software may be easy for you but wiring up circuits may be another challenge. Fortunately there is an IDE called Frizing that helps you design your breadboards. You have to check out this great tool! Figure 1. shows the toolkit that you can used to place parts on the virtual breadboard that Frizing provides. Then the tool can be used to generate a schematic diagram. A newcomer to Do-It-Yourself techniques may not be familiar with all of these terms. We will be using Fritzing breadboard diagrams throughout this book however we may use schematic diagrams produced by another tool, Kicad as they look a lot better.



MIT Scratch

If you are looking for an introduction to programming that children might understand you might look at MIT Scratch. This is an outstanding visual approach to programming that uses an anime to explain what is going ton. The Maker Robot actually uses this tool as their main programming environment. You can program the Arduino with this tool as well.

GNU Eclipse

So many Integrated Development Environments for embedded development are based on GNU (which stands for GNU is Not Unix) that it is good to know how to download and use the free Eclipse IDE and associated tools oneself. Once you are up-and-running with the latest version of Eclipse on your laptop or desktop you can add plugins

that can help you with your project and build up a customized version of Eclipse. Of course many small computer boards and systems come with customized versions of Eclipse just like the different versions available for some programming languages. One of these customized versions of Eclipse for Arduino is Sloeber (eclipse.baeyens.it) which looks like a really great way to look under the hood of Arduino Code.

TI CCStudio

CCStudio by Texas Instruments is another example of a now free IDE that can be used to program the TI MSP430 and DSP processors.

ARM IDEs

Integrated Development Environments (IDE) used to program the popular series of ARM chips, such as IAR Systems and Keil Systems are expensive. You don't really need them to get the ARM-based Raspberry Pi, which is a full Linux system with a kernel already included, programmed, but for some embedded controllers based on ARM technology such as those offered by ST Microelectronics you will need one of these programs.

It is somewhat interesting to note that while the computer hobbyist uses a free IDE to create programs, line-by-line, for the Arduino and Texas Instruments chipsets, when it comes to programming the ARM boards one uses an embedded operating system that provides programming languages on-board as we will see in the later chapters on Operating Systems and Rebuilding the Kernel.

3 A First Embedded Program

The first project most people try with their small board computer is Blink, a program that blinks a Light Emitting Diode (LED). On the Arduino UNO it is easiest to use the Arduino IDE to blink the onboard LED on port 13. Whoa! Stop right there! All of this needs a bit of explaining.

It may seem futile to use computers, which have reached such an advanced state of technology that a typical Intel i7 desktop may have 4 64-bit cores, to blink a light emitting diode but we sometimes forget that each one of the bits in a computer corresponds to encoded charges, and travels on wires however microscopic.

So to use the Arduino a 8-bit processor to turn one line on and off and light a light-emitting diode really brings us back to reality.

Just try getting your personal computer or laptop to do this. Now do you understand what this is all about? Capisci!

Perhaps you are only slightly familiar with computer programming. We have purchased our first Arduino UNO or compatible board. What do we do next? If we were learning to use a new programming language we might try to write "Hello World!" to the console or terminal. A C language program to do this might look as follows:

```
#include <stdio.h>

main()
{
     printf("Hello World!\n");
```

```
}
```

In this program, after the "libraries," chiefly the standard input and output library are included via its "header file," the program code within the braces of the "main" function are run to print out "Hello World!"

In a similar fashion we write a program to blink an "on-board" LED on the Arudino UNO at the rate of about a second. The light of the LED goes on for a second and then it goes off for another second continually. Here is the code to do this:

```
setup()
{
     pinMode(13,OUPUT);
     digitalWrite(13,LOW);
}
loop()
{
     digitalWrite(13,HIGH);
     delay(1000);
     digitalWrite(13,LOW);
     delay(1000);
}
```

In this "main" program, two function blocks are executed. The program looks like another C program but it is really Arduino C++ code. Both setup and loop are functional blocks that are completed as part of an underlying class (look at Beryllium and find out what class). In the 'setup' routine the pin of the Arduino Uno used to control the onboard LED, or pin 13, is set to ouput mode. Then a 'LOW' or zero value is written to it, causing the LED to start out in the OFF state. In the loop which runs continually the LED is brought HIGH or on, the program waits for 1000 milliseconds or one secont, the LED is turned off and it waits for a second again before repeating itself.

An Introduction to C++

Arduino and many other embedded computer boards are programmed using C++. Node.js and Circuit Python are also used. But

perhaps the quintessential 'low-level' language is C because it almost parallels the CPU internal opcode or assembler language structure; this does not make it a better programming lanuage to use for embedded applications. Of all the programming languages that are used in embedded development, C++ is possibly the most graceful of the all in that, like C, it does not "encapsulate" the details of the machine as much as node.js and circuit python might, while still providing class-oriented programming.

In C++ objects that are physical code elements in the operating machine are "instantiated" from classes. One might be tempted to say that objects are a superset of classes. But it is better to think of classes as the templates for the objects that will be in the system. The class is the progam code that tells the instantiated objects, created from that class, what instructions to follow during the time that they are running. We can liken this process to the procedural language (C) equivalent of the main loop being loaded into memory and run. When a C++ object is loaded into memory the instructions in the class are followed to tell it what to do.

Abstracting Blink using a C++ Class

We can abstract the blinking LED into a class of its own so that the code is more clear. This enhancement to Blink will give us a huge jump in clarity if not functionality as well. We have added two new files to our program to accomplish this. The first is BlinkLED.h which is the C++ "header" file:

```
// Listing of BlinkLED.h
#include <Arduino.h>

class BlinkLED
{
    public:
     BlinkLED::BlinkLED(int pinNumber);
     void BlinkLED::blinklight();
    private:
     int ledPin;
             int status;
};
```

There are several things to notice about this file, and for the beginner to programming we note that the syntax of a program is just about the best place to begin learning about a new programming language. First this file, which we call BlinkLED.h defines a class called BlinkLED. The objects instantated from this class will have two public methods, the constructor "BlinkLED" and a function, blinklight. The constructor is a special method that starts the new object running. Our new blink program will call the blinklight() function everytime it needs to change the state of the LED from on to off or off to on. The two private variables pinNumber and state in the class definition (header file) will be used in the program of the running object to keep track of what is happening:

```
// Listing of BlinkLED.cpp
#include <Arduino.h>
#include <BlinkLED.h>

BlinkLED::BlinkLED(int pinMode)
{
    this->pinMode = pinMode;
    this->state = 0;
}
void BlinkLED::blinklight()
{
    if ($this->state != 0)
    {
     $this->state = 0;
    }
    else
    {
     $this->state = 1;
    }
    digitalWrite(this->pinMode,$this->state);
}
```

Don't worry if you don't understand everything about this code snippet. It takes time to learn new features of the C++ syntax. But please notice that the public and private members of the BlinkLED class

that are defined in the header file previously are used here. What remains is to use this new class in our new Blink program: ClassBlink.ino:

```
// ClassBlink.ino
#include <BlinkLED.h>

BlinkLED Blink13;

setup()
{
     Blink13 = BlinkLED(13);
}
loop()
{
     Blink13.blinklight();
     delay(1000);
}
```

If you notice one thing about ClassBlink.cpp it might be just how much simpler the main program is to read and understand, and in fact, execute. It was a lot of work to "encapsulate" the "functionality" that we previously had encoded into the main Blink program earlier into this class but now we have a BlinkLED class that could actually become a "library" if need be. We are just a step away from understanding what C++ Libraries are and how to use them.

The Processor 4

The vast array of processors to choose from is bewildering. One might employ a classification system of sorts to choose between them. Depending upon the project, one chooses a different processor and small board computer. Although using one variety or brand of processor for all of one's projects may be easiest in terms of mastering the learning curve, there may be cases where another processor is better for the application. In this chapter we look at the different processors and boards and select some for our projects.

Some of the factors that we use to choose a processor are throughput, system requirements, input / output and memory requirements and cost. Software availability and ease of programming are also critical to the successful completion of an IoT project. If a chosen processor works with some of the IDEs we reviewed in Chapter One all the better.

For example that the TI MSP chip might be best for the Inventory control arm while Arduino is better for the weather station, raspberry pi for the linux like things and x86 for stuff that requires main stream software packages and data processing.

It is one thing to choose an processor, and quite another to have running software for it. The second part of this chapter looks at some of the operating systems and toolkits and discusses their interoperability or lack of it.

What is a Central Processing Unit?

I have always been helped in my work in computers by having a basic understanding of the central processing unit (CPU). Once one

has learned digital electronics it is a fast progression from binary addition to understanding what is an Arithmetic Logic Unit (ALU) and the registers and control lines of the processor. It is not my goal to review all of that material in these pages, however for the purposes of building edge computers it is helpful to have the computer architecture in mind.

Microcontrollers vs. Microprocessors

There is difference between a Microprocessor Unit (MPU) and a Microcontroller Unit (MCU). Both devices are monolithic integrated circuits or Central Processor Units (CPUs) on a single silicon chip. These are the precursors to the modern Systems-on-a-Chip (SoC) that underlie most of the new processors and devices. According to Merriam Webster's computer dictionary a Microprocessor is:

"a computer processor contained on an integrated-circuit chip also : such a processor with memory and associated circuits"

A microprocessors is a single computer unit that must be made into a computer with external chips and devices whereas a microcontroller already has some of the input and output devices needed for IoT.

RISC vs. CISC

Terminology such as the "clock cycle" may be new to the beginner in IoT development but they are useful to understanding some of the different sorts processors that are available. Suffice it to say that the newer processors such as ARM processors that are used for the Internet of Things complete all of their instructions (opcodes) in a single clock cycle. This is called Reduced Instruction Set Computing (RISC). Programs in a RISC computer can more accurately respond to real-time events because the programs are guaranteed made of atomic, that is integral or countable, operations.

Older computers (desktops and laptops) used chips that were based on Complex Instruction Set Computing. A given instruction (opcode) could require any number of clock cycles to complete. Suffice it to say that one might read further to better understand the implica-

tions of these design choices.

The Harvard Architecture

The newer chips that are used by IoT also employ what is called "Harvard Architecture." Chipsets such as the ATMEL AVR processors store their programs separately from the data on which they perform computations. Recall that Turing pioneered the concept for computer theory of the infinite tape in which a tape head constantly read instructions off of a linear tape and performed simple actions accordingly. The microprocessors of the 1980s and 1990s read opcodes from the same data bus used to send and receive the results of computations. Although this architecture may have fit hand-in-hand with CISC processors that needed many clock cycles to complete calculations one the program counter (register) relinquished the use of the data bus, today reduced instruction set computing works better in a real-time world.

What is the Arduino Microcontroller?

Arduino is an AVR Microprocessor invented by Massimo Banzi in Italy. It is a very popular microcontroller.

AVR Microprocessors are Harvard-architecture RISC-based semiconductors sold by Atmel, a Norweigian company.

Arduino is a registered trademark for a small board computer based upon these chips. It is open-source electronics here, but in Italy the open source version of this board is called Genuino. I get confused when I read the history of who sold what to whom and how the trademarks came into being.

Suffice it to say that the Arduino standard has become so popular that many other chipmakers support this standard. The Arduino UNO that we have in this course has about twenty five pins on "header" sockets that are used to support shields. Arduino shields have rapidly become a standard.

That's not all that the Arduino toolkit has shared with other man-

ufacturers. Intel and many more vendors have created computer boards that use the Arduino "form factor" and the Arduino IDE (referenced in Chapter Two) to program their own boards. For a while, Intel was selling boards that they called Arduino 101. But even though that board did not become popular and was retired, Intel still sells a D2000 Development Kit. Even Raspberry Pi boards with the right "hat" can be used with Arduino "shields" or daughter-boards.

Once we choose to use an Arduino or Arduino-compatible board we must still decide which one we are going to use. No matter what we will soon try writing our first program, which is to blink the on-board LED.

What Small Board Computer Should I Use?

You may not have heard about Raspberry Pi. But if you are just signing on to the Internet of Things trend it is not too late to find out more about this device. I am stubborn however, so I usually like to begin shopping for any kind of device by looking at the alternatives! There are a lot of alternatives to Raspberry Pi like Orange Pi and Banana Pi and ASUS Tinker Board. These are all ARM devices. Don't ask me too much about that; but I have learned that ARM is the new standard in the world of small and tiny computers. ARM computers are what are known as Reduced Instruction Set Computing that are better for low power and fast on-the-fly computations than the computers of the past.

The days when my classmates chattered excitedly about building the next Motorola or Intel based computer are long gone! What have we just stumbled into? A world of experimenters' wiring and chips and technologies mastered by only an obscure few? Although that is part of the mystique of the Internet of Things there are a lot more ways in which people can participate with this technology. For the tiniest systems you need ATMEL; the Atmel 328 and the Arduino chipset or the ESP8266, which comes with Wifi, or Texas Instruments' MSP430. If you are like me however, having just gotten through the iPhone and Android application craze without much success you don't need any more information! But if you are in fashion you might want to wire an Arduino chipset like the LilyPad to a LED (Light Emitting Diode) lapel pin.

Remember that we are all on the Internet now! It makes sense to be asking how we can be more competitive in our small-town world. Maybe it doesn't make sense to have a refrigerator that can ask "Got Milk?" but it might be reasonable for our smart kitchen to compare prices at stores or suggest improvements to our usual recipes based upon what others are doing out on the Internet. Here is where we may wish to look at Gumstix, Inc's device for communicating with the Alexa API. This notion is itself the new method of the new web-enabled world. In our daily lives, if we are smart, we are interacting with more and more people than ever, on our cellphone, on our computers, and now with the things we love to do; like our jogging clothes. Our smart devices are helping us learn from others and redraw a more prosperous map of the world.

So, build your own smart devices. It's good to be using what is commonly available. Raspberry Pi and Arduino are solid boards and are now standards. I would be interested in looking at the UP board with onboard FPGA to really take a leap into the future.

Vendor Loyalty

Before you tell me that I am unfairly taking sides a new small board computer debate (we used to call this the Programmer's Language Shootout at the Amateur Computer Group of New Jersey - ACGNJ), let me remind you that I am the guy who spoke in 2016 about x86 embedded development boards. Intel's proprietary x86 opcode standard is still around, even within the Internet of Things communities. Don't be fooled into thinking that ARM or Raspberry Pi is the only board around or the one you must use. You must decide what board is best for your project.

Like programming languages, different computers are good for different things. But comparing computer programming languages is not quite the same as comparing computers. We can use different programming languages to program the same program, such as "Hello World" or a test to see if a number is a Prime Number. When we are through with our comparison it still may not be such an important matter if we use Python or PHP, an interpreter or a compiler. Regardless of what programming language we choose the program must run on the computer hardware that is provided. Our

choice of programming language has more to do with what we want to accomplish than how we expect to get it to run.

Choosing computer hardware and processors is a little different, especially for the Internet of Things. Some computers just aren't powerful enough for certain applications even if they are programmed to do so. Then we must think about a more robust CPU – Central Processing Unit. Other applications, such as signal processing might need a RISC – Reduced Instruction Set – machine to operate fast enough. Our choice of CPU can influence what programming language we might choose, or it might not! We must not always be hardware or programming language agnostic.

Processors

The 'workhorse' of these Internet of Things devices is the processor. We choose to use the colloquial word 'workhorse' meaning engine rather than stating that the processor is the brains of the machine. These computers and processors are electronic machines that have been programmed to think. But what is this "thought" that computer processing engines might be said to possess?

Computers are essentially digital calculation machines and follow a set of instructions to do various binary functions such as adding, subtracting, shifting bits, and comparing values. These basic calculations run frequently and repeatedly and allow the computer to compute values and make decisions that can change the outcome of a device depending upon conditions.

Internet of Things devices all acquire data from the signals in their surroundings. There are two ways that embedded processors can obtain signals: through digital inputs and analog-to-digital conversion.

Arduino

As you can imagine, getting small embedded computers that are really nothing more than chips on a board to operate properly can be quite a task. The Arduino boards (Arduino.cc) are a popular solution that allow one to get started with the Internet of Things with just a

laptop and a USB cable. The most basic Arduino board, the Arduino UNO, has 13 digital inputs and outputs and 6 analog inputs.

Arduino Variants

Because the Arduino design is an open source standard, there are a lot of other boards that are programmed to use this specification.

Raspberry Pi

Like Arduino, Raspberry Pi has become another standard in the Internet of Things. This is also a small board computer with input and output lines, but unlike Arduino, which runs a dedicated program in its programming space, Raspberry Pi runs a full-fledged operating system such as Linux or Windows.

Raspberry Pi Variants

Variants on Raspberry Pi include Banana Pi, Orange Pi and others. The above mentioned small board computers all use ARM technology. ARM is a Reduced Instruction Set Computing chip that crunches numbers a lot faster than the older Intel x86 Complex Insruction Set Computational chips. Nonetheless these x86 chipsets do a lot more per cycle than the ARM chips. Also there is a lot more software that the older Intel chipsets can run.

Arduino 101

The Internet of Things is so popular that Intel even introduced its own version of Arduino called Arduino 101. Although they have discontinued this chipset, based on the Intel Curie, boards like UDOO board have an Arduino 101 on-board. At this point it appears that ARM is the clear winner in the IoT competition. Nonetheless Intel x86 is not going away. You can still buy your own brand new D2000 MCU directly from Mouser

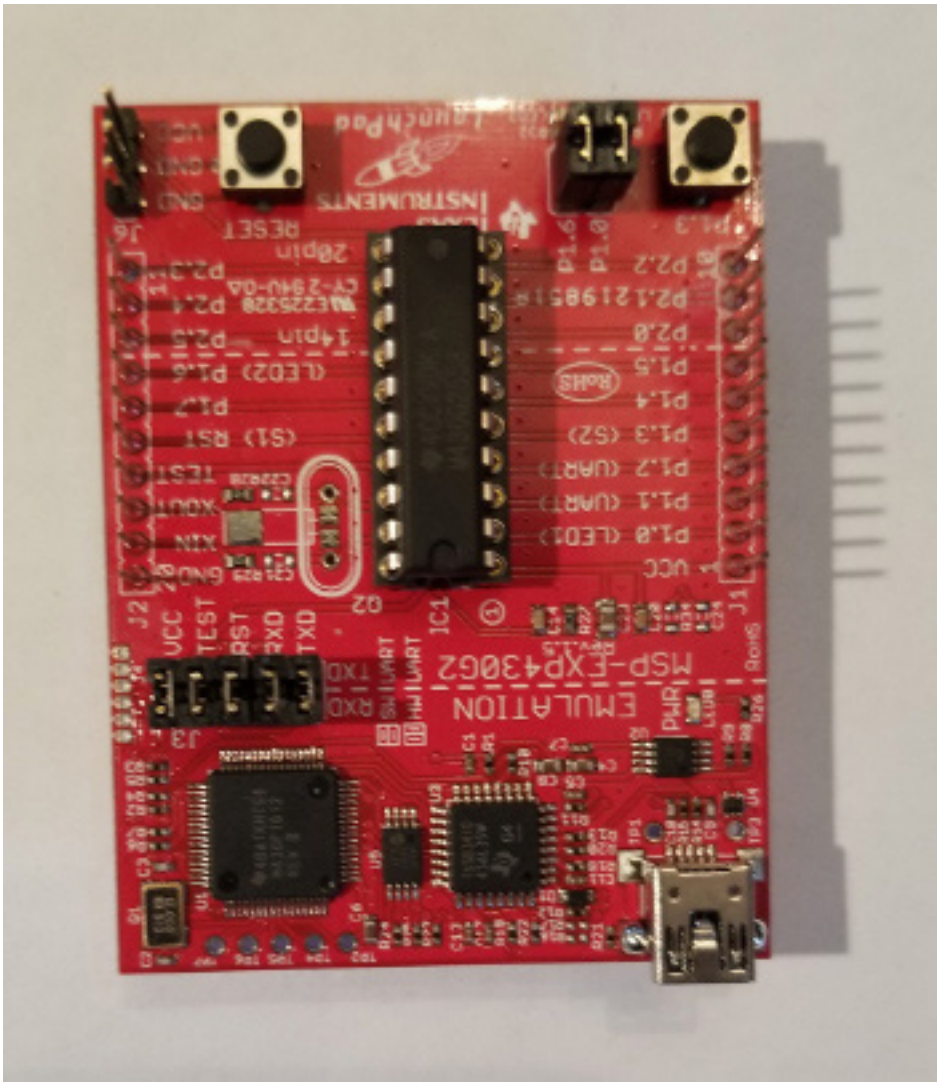
Other Intel x86 Boards

UP Board, UDOO Board and Latte Panda are all Intel x86-based boards. Like Raspberry Pi they can run Linux but they can also run Microsoft Windows CE. It's pretty amazing to think of running Windows on such a small computer.

Texas Instruments' Embedded MSP430

Of course, if you are going to shop for computer chipsets be sure to look at TI's MSP430 LaunchPad. These devices have a smaller footprint than the already small Arduino.

And TI has gone to great lengths to ensure that the programming toolkit for these boards are readily available. The IDE is easy to use and the software more accessible from the command line than Arduino.



FPGA

At some point we might ask if there is a better way to build end user devices than using all of these embedded processors, with all the overhead required to operate them. This brings us back to where we started this chapter when we said we might not even want to use a processor for some of these reasons. Field Programmable Gate Arrays are a way to create circuits that are customized for a given application. There is a lot of overhead to program these too, but you can't let that scare you.

Programming the Internet of Things

We have spent a lot of time talking in these pages about embedded processors and small board computers and their use on the Internet of Things but we haven't built a single thing that can interact with the Internet. In the next chapter we take a new look at the standard program used to control a blinking Light Emitting Diode (LED).

5

An Arduino Thermometer

The Internet of Things is a great new networking tool in our toolkit. We must take advantage of this new tool! What can we build with the Internet of Things? Knowing what we are going to build will help us to outline the different components and subsystems of an Internet of Things device.

There are a lot of details that must be considered to build a device based upon the Internet of Things. Throughout this book I will be giving you examples that you can use yourself in your own projects. These projects are the real building blocks of the Internet of Things. Like running a test "Hello World" program in the programming language of your choice, these projects help to establish the pieces that can make up our next Internet of Things installation.

As we think of the projects we would like to build, a remote weather station, a water-level warning system, a telephone, and other innovations, if not brand-new inventions, we realize that these are simply electronic devices and not particularly enabled for the Internet of Things. These devices must also communicate over the Internet or at least some network to be considered Internet of Things devices. It seems too obvious to state, but we must have the building blocks in place, the working basic components in place before we can build out.

As regards the weather station, we might want to start by building a regular electronic temperature reader so that we understand the basic functionality of our final design. The readings, such as the air

speed and temperature that this station is supposed to measure are shown in the diagram. There is no point in designing the perfect weather station if there is no way for it to send out the data. Nonetheless, starting with a non-IoT device might make the design process simpler.

We ask the following questions: What will these devices do? How will they operate? How will they gather data? How will they transmit that data "upstream?" What components will we use to prototype and later build them? What construction techniques should we use? What processors and communication chipsets should we choose?

Technology should always look forward, so it is alright to take a look at what has already been done. We don't necessarily need to "go retro" to have a look at where technology was a decade or more ago. And although we wish to move forward with new technology we maintain very many old conventions. For example we may use FPGA modules. We might run an older version of Windows within a Virtual Machine. Or we might use a new technology like Docker, which are themselves a reuse of Linux Containers to develop our programs more independently of the users's hardware.

As we rebuild devices that have been invented before – "reinventing the wheel" -- we must embrace not only the Internet of Things and design these devices to be compatible with IoT but also try to fulfill the new requirements of recycling, repurposing and reusing that are such a part of the Maker and Mender movements that we described in the last chapter. Why not reuse or repurpose old designs as well?

An Indoor / Outdoor Thermometer

As a first example of an Internet of Things project, consider an Indoor/ Outdoor Thermometer. This weather station is to report back on its surroundings. We want this device to be on the Internet of Things so that others can remotely view the weather conditions at our location.

The DHT22 Temperature and Humidity Sensor makes a great way to build a digital thermometer. Before we can put data from a device such as a weather station onto the Internet for use by others who

will read the data using the Internet of Things, the weather station itself must be built. In this Chapter we will build a basic Arduino device for reading the temperature and humidity and display the results on an LCD panel. Figure 2-2 shows a very basic block diagram describing the operation of this device. In a later chapter on Bluetooth and Wireless connectivity we will describe how the data from such a unit can be sent upstream.

The Path to the Internet of Things

A block diagram for this project, shown in Figure 1, shows a Micro Controller Unit reading results from the temperature sensors and sending these values out over the network. The MCU is central to the operation of this device. But reading the temperature doesn't have to be that complicated. One can instead utilize a thermocouple or an IC that sends an analog signal back to the rest of the circuit. This analog signal can be converted to digital or used as is. The solution chosen depends upon the requirements and need for this device.

Think for just a bit about the layer of complexity added by our requirement that the thermometer and barometer in this station report values across the Internet. No longer can we utilize a stand-alone electronic or mechanical device. Extra power requirements are created by the device being a part of the Internet of Things. If we are smart however, we might see if another solution might fit our needs. Then too we can ask if the compute power provided by an IoT device can help us to better create the device.

Is a Processor Needed?

The answer to the question "Is a processor needed for an Internet of Things building block?" is "yes" but the answer may not be obvious. A given circuit to measure temperature might read analog electric signals from a thermocouple for example. Depending on how it is needed, this reading could remain in an analog format for entire data cycle of the device. Imagine for instance that the thermometer temperature value was beamed in infrared light to a receiver which displayed the reading on an analog meter. (the building block of the analog meter ... has it been forgotten). There might be no computer involved at all.

An Analog Electronic Thermometer

We consider an analog meter first. Since this meter is used to tell the temperature outside while we are comfortable in our house we may only need a scale we can read from the window.

Humans themselves can tell if it is hot or cold but it is hard to know exactly what temperature it is outside. By the window to our kitchen next to the bird feeder we can mount an outdoor thermometer. This may be a standard glass bulb containing a liquid that expands as the temperature rises; indicating the exact temperature on a calibrated scale. If that is all that we need then we are in good shape.

The next step in our odyssey is to use an electronic sensor. A thermocouple might be the best way for us to tell the outside temperature. We must find some way to transmit the electrical value read by our sensor to a front panel. If wires are too cumbersome, we might choose an infrared link to a unit containing a screen where we can read the value of the temperature.

The bottom line is that we must often remember to ask is "What are we trying to accomplish?" Why is a processor needed? What does a digital processor do in a circuit? Let's continue with the above example. How is the measurement going to be transmitted over infrared light if we don't convert to digital?

Depending on the temperature (voltage level in the wire) we can blink an infared light at a faster or slower rate. Then on the receiving end we can report the temperature on the far side of the room by counting these blinks. If we were to replace this circuity with digital then suddenly we would have a computer; a device containing instructions run repeatedly counting up these blinks of light and displaying them on a screen.

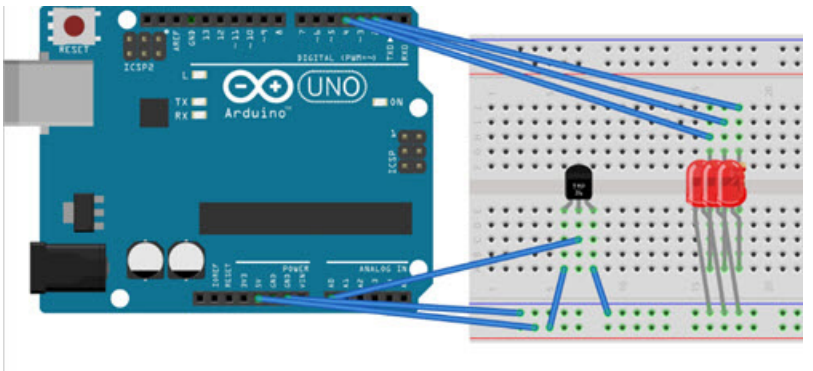
Like the examples in Chapter Two this example may seem to be contrived. But a lot of computation and analysis can be accomplished with analog electronic and even analog computers. When we move to the digital equivalent of these circuits we may need to choose a processor capable of handling very fast computations that can take

the place of these analog circuits, like TI's DSP boards or Red Pitaya.

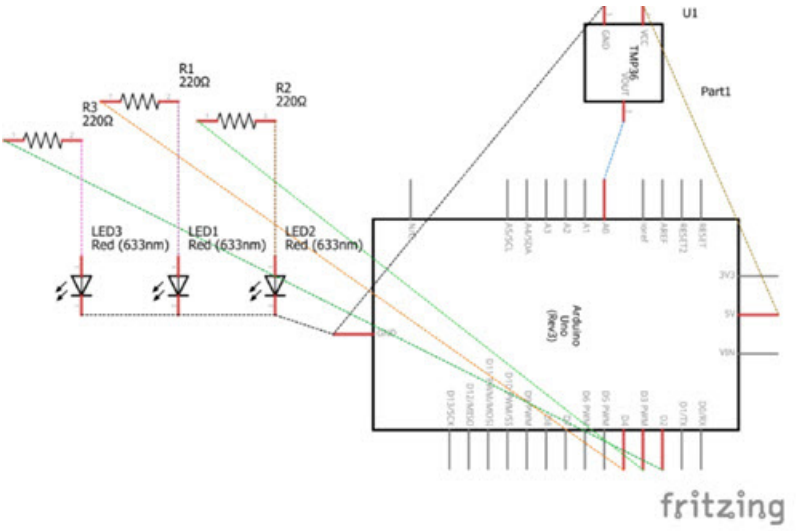
The point of this section isn't to force the reader to take a step back on the way to the Internet of Things. It is to note that before digital computers were so embedded there were a lot of ways in which analog circuitry could accomplish the same role. These techniques have become important again to enable the Internet of Things. The Internet of Things revolution is possible because on every level, digital computational devices can modify the behavior of these basic electronics building blocks as a result of an external input – that is by being reprogrammable, be somewhat smart.

A Digital Thermometer

We start building a digital version of the thermometer at one step beyond the thermocouple or thermistor.



Here is the schematic for this circuit:



A Fancy Solution

When we are ready to digitize the value of temperature and send it to another computer we have several options. We can feed the result of the above circuit into an analog-to-digital converter and monitor the temperature. Or we can use a digital sensor such at the DHT22 device.

The DHT22 sensor, which contains both temperature and humidity sensors is perfect for this application. A small device with three wires that communicate back to an edge computer with the I2C bus (what is a Bus? What is I2C?) it only represents perhaps several solutions to this challenge.

Since the Arduino does not contain Internet like the Raspberry Pi we will use the HC-05 Bluetooth module to send the data to another computer. Then we will be ready for the Internet of Things We have built an excellent Arduino thermometer but have never connected it to the Internet of Things. One way that we can read the temperature at some distance from our newly-built device is to use the Blynk mobile phone application. This application is aptly named because the first program that many people to use to test the operation of their

devices is to blink a light on the computer board, the Blink program as it is known to Arduino enthusiasts. Figure 2-3 shows a picture of the Blynk control panel.

Temperature Over I2C

The first step in getting the Arduino thermometer to operate is to try the demonstration example that comes with the DHT-22. The DHT-22 is a sensor having three wires. How hard could that be to wire up? We don't really need a Fritzing diagram to show this so we have presented the schematic instead.

The one data line that we have connected to the Arduino Uno board on pin 7 is an I2C line. This is a "one-wire" communication protocol for communicating digital values. Our next step after building this circuit is to uplad the example sketch. Then we can used the Arduino's serial monitor to see the temperature and humidity.

Avoiding Conflicts on the Bus

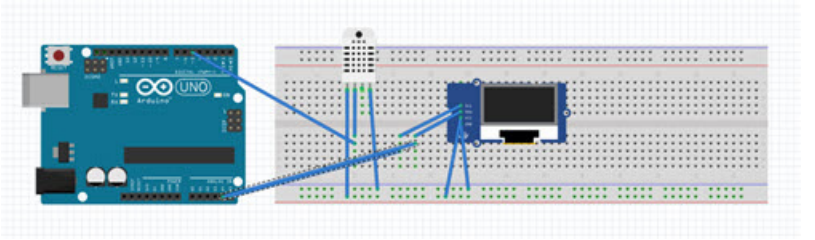
Getting the DHT-22 sensor to work on its own is easy enough. But we also hoped to display the temperature and humidity values on a small OLED display. As one adds more components the circuit complexity increases. One must understand both the timing and port requirements of all the components in a circuit in order to have them work together.

Before and After the Internet of Things

Before we had the Internet of Things we could only locally tell the temperature. It was an improvement to move from analog to digital devices but not nearly as big a step as enabling a thermometer to report its temperature to an Internet-enabled device. With the Internet of Things we can read the temperature on our cellphone. With IoT people far away from us can monitor the weather conditions at our location.

Internet of Things Matrix

The first thing one notices is just how much processing power is required by a Central Processing Unit or Microcontroller to accomplish the tasks that are outlined. Given the importance of the processor, this chapter will discuss this central component.

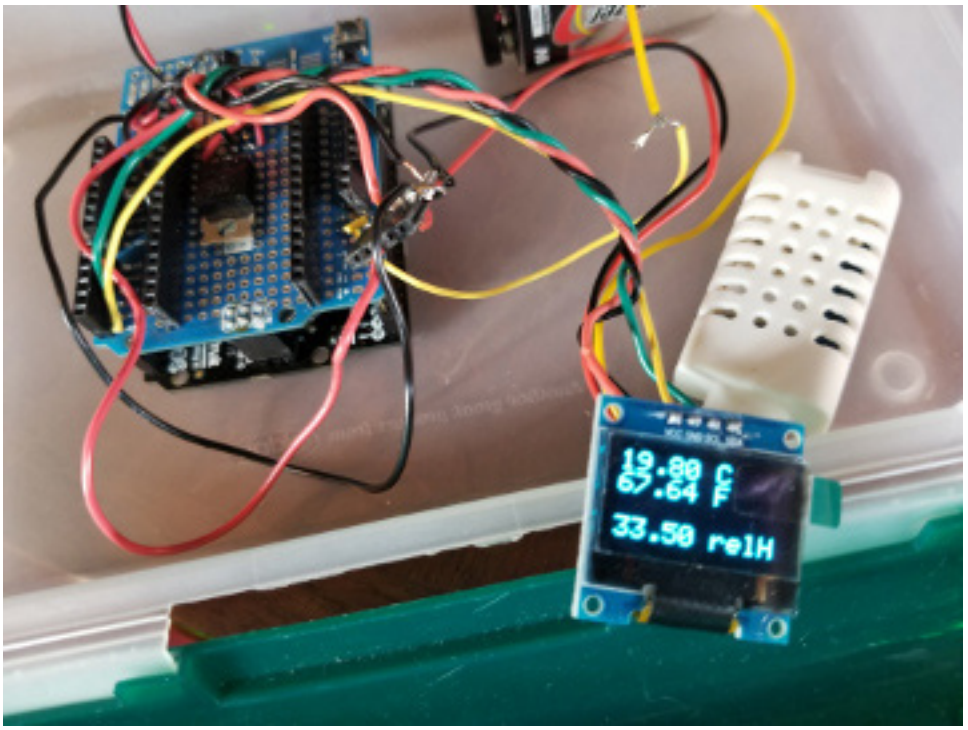


Device Behavior

Since this book discusses the requirements of Internet of Things devices and the software (also a building block) that is necessary to make them act in a smart way (much less operate at all), it also provides a way for developers of software and other devices to the model behavior of these devices. The role that the device plays in our environment has changed from before the Internet to now. Who is modeling the IoT environment?

The Final Design

Finally, we have understood what our Internet of Things device is to accomplish. We know what it is going to do, if it does anything, and we know what data it will acquire. We have some notion now of how the device will need to send that data somewhere and respond to both changes in its environment and commands. Now we are ready to look at the small board computer and processor that will be needed to make the system run.



Following is the Arduino code that we used to get our stand-alone device to run.

```
// DHT Temperature & Humidity Sensor
// Unified Sensor Library Example
// Written by Tony DiCola for Adafruit Indus-
tries
// Released under an MIT license.

// Depends on the following Arduino libraries:
// - Adafruit Unified Sensor Library: https://
github.com/adafruit/Adafruit_Sensor
// - DHT Sensor Library: https://github.com/ada-
fruit/DHT-sensor-library


#include <Adafruit_Sensor.h>
```

```
#include <DHT.h>
#include <DHT_U.h>
#include <Arduino.h>
#include <Adafruit_SSD1306.h>

#ifdef U8X8_HAVE_HW_SPI
#include <SPI.h>
#endif
#ifdef U8X8_HAVE_HW_I2C
#include <Wire.h>
#endif

#define DHTPIN            5         // Pin which
is connected to the DHT sensor.

int count = 0;
int toggle = 0;

// Uncomment the type of sensor in use:
//#define DHTTYPE           DHT11     // DHT 11
#define DHTTYPE           DHT22     // DHT 22
(AM2302)
//#define DHTTYPE           DHT21     // DHT 21
(AM2301)

// See guide for details on sensor wiring and
usage:
//   https://learn.adafruit.com/dht/overview

#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);
//display_SSD1306_128X64_NONAME_F_4W_SW_SPI dis-
play(display_R0, /* clock=*/ 13, /* data=*/ 11,
/* cs=*/ 10, /* dc=*/ 9, /* reset=*/ 8);
DHT_Unified dht(DHTPIN, DHTTYPE);

const long screenInterval = 1000;           //
interval at which to buffer (milliseconds)
unsigned long sensorInterval;
unsigned long previousMillis;
```

```
void setup() {
  pinMode(13,OUTPUT);
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
// initialize with the I2C addr 0x3D (for the
128x64)
  dht.begin();
  // Print temperature sensor details.
  sensor_t sensor;
  dht.temperature().getSensor(&sensor);
  // Set delay between sensor readings based on
sensor details.
 sensorInterval = sensor.min_delay / 10000;
    }
void loop() {
        char buffer[50];
        bool isTemperature;
        bool isHumidity;
        float temperature;
        float fahrenheit;
        float humidity;

        // Get temperature event and print its
value.

        sensors_event_t event;
        dht.temperature().getEvent(&event);
         display.clearDisplay();
  // Get temperature event and print its value.
  display.setTextSize(2);
  display.setCursor(0,0);
  display.setTextColor(WHITE);
  if (isnan(event.temperature)) {
    display.println("ERROR");
  }
  else {
    temperature = (float) event.temperature;

    display.print(temperature);
    sprintf(buffer," C");
    display.println(buffer);
    fahrenheit = (9 * temperature / 5) + 32;
```

```
    display.print(fahrenheit);
    sprintf(buffer," F");
    display.println(buffer);
  }
    display.println("");


        // Get humidity event and print its val-
ue.
        dht.humidity().getEvent(&event);
        if (isnan(event.relative_humidity)) {
          isHumidity = false;
              display.println("ERROR");

         }
         else {
           isHumidity = true;
           humidity = (float) event.relative_hu-
midity;
    display.print(humidity);
    sprintf(buffer," relH");
    display.println(buffer);
         }
         delay(sensorInterval);
  display.display();

}
```

6

Communication Protocols

When we speak of Protocols what do we mean? Although we are especially speaking of standards for communications when we talk of protocols many of the sensors such as the DHT22 Humidity and Temperature Sensor that we saw in Chapter 5 use protocols as well. When we speak of the protocols used by Internet of Things and other devices for communication we must describe the transport medium, the hardware and the supporting software. We must write our programs for our devices to be able to communicate using the proper protocol on these media.

Universal Serial Bus

Sometimes one must explain things more clearly. I learned how to do this in 2008 when I consulted a Mathematician about my paper on Tree Lattices. To explain what he said let's take USB as an example. Consulting Wikipedia https://en.wikipedia.org/wiki/USB I find that USB stands for Universal Serial Bus. In order to explain the Universal Serial Bus I must first describe what is a Bus, what are Serial Communications and describe the improvements to the Serial Bus that enable the Universal Serial Bus.

What is a Bus?

A bus is a bunch of wires used to transmit digital information. If we speak of a serial bus, we are talking about one or two wires in which data is constantly transmitted, one bit at a time. The alternative or opposite of a serial bus is a parallel bus. In a parallel bus, many

wires, say 8 or 16 are used to transmit an entire byte or word at a time to a device. In digital electronics each wire transmits one of two binary values, high or low, on or off. If only one wire is available a byte of eight bits must be transmitted one after another similar to Morse code.

The Communication Ports or COM Ports

Standard serial connections used to have four or five lines and were called COM ports. These wires were Ground, 5V power, transmit, receive and clear-to-send, perhaps. There were a lot of different variations but they all worked in this way. Data was sent in a serial stream of bits over the transmit line and received by the receive line. Serial ports were more useful than parallel ports for transmitting data from a computer to a modem (to send data out on a conventional telephone line) but they could be hard to troubleshoot.

The Universial Serial Bus, Take Two

When the Universal Serial Bus or USB was introduced it rapidly became a new standard for computer communication. Although it uses two data wires to transmit information, like the serial port, the USB protocol includes a lot more control sequences. But, whether your system uses a USB controller or outputs serial data using the older UART (Universal Asynchronous Receiver / Transmitter) chips, both sides of the transmission need to be able to convert this data back into something meaningful. For Arduino projects it is helpful to use a USB-to-UART convertor to provide serial communications from a modern computer.

I2C, SPI and Others

There are a lot of other protocols that use serial comminications that are especially good for the Internet of Things. These include the I2C and SPI busses which use serial communications to transmit data to and from devices rather than whole systems. The DHT Sensor used "One Wire" to receive digitized temperature information while the OLED display that we used required the I2C bus.

TCP / IP

Computer networking protocols such as TCP / IP, Transmission Control Protocol / Internet Protocol, and UDP /IP, Universal Datagram Protocol / Internet Protocol are important for the Internet of Things as well. This is the standard that underlies communication over the Internet between computers and communities. It is also the basis by which IoT devices can communicate with each other and may probably be a standard for commications with 5G and LoRaWAN radio links.

Whatever mneumonic you use to remember that the seven layers of the OSI (Open Source Internationial) reference stack are Physical, Data Link, Transport, Session, Presentation and Application (did we miss one?) sooner or later even a small device such as Arduino may use this form of communcation to send and receive data.

7 Sensors and Devices

Most of the IoT projects we hope to build require real world inputs and outputs: sensors and motors and electronic devices of all sorts. So far we have delved deeply into Arduino development for projects that are best suited to the low level, real world control afforded by the AVR processor and the ATMEL chipset. The fact is that some of the other projects we may want to build on different platforms may also need to acquire data from the environment.

You may wonder why we may suddenly wish to switch processor type and vendor once we have already found a standard like Arduino that serves us so well. As we mentioned in Chapter Three, a different processor type or processor architecture may be better suited for certain applications. We may want to use a different Atmel processor with more GPIO pins or a faster clock rate. Or when we start to look at processors that are more suitable for some of our larger IoT projects, such as the Intel boards that might run an operating system capable of managing an inventory for an inventory robot we hope to build, we will still need real world inputs and outputs: to operate robotic arms, respond to limit switches or use computer vision to inspect our work.

But the Arduino Blink program certainly made life easy. I have to tell you a funny story. When I was at Princeton University, my professor asked me to use MSDOS to analyze a combustion engine. But no matter how I tried, I could not dive through the layers of complexity required to even turn on a light emitting diode. It may have been just an 'out()' function call statement in a C program that I needed but I soon became lost in pages of device driver material. Now I am good at that stuff.

Years later I met someone at a Princeton Reunions who succeeded at completing this assignment after I had failed at it! I've learned to be more careful in using Application Programming Interfaces (APIs) since then.

The final project in this book, the Internet of Things Frisbee is another device that might be better suited to a different processor than the Arduino but will require sensors to enable it to operate. Right now, I am looking at a Raspberry Pi-variant, the ARM-based Khadas board with its on-board wifi to fly in the frisbee. Additionally I am going to need to 3d-print a custom frisbee. The frisbee will have to contain battery power and have an onboard altimiter and gyroscope.

There are many categories of sensors that can be used for IoT projects. We can group these roughly into four or five categories. The mechanical actuators are a separate topic.

Switches and Buttons

Perhaps the simplest kind of input for Internet of Things devices is the electrical switch or button. These allow us to give a digital input, on or off, into our devices. An electrical button may need to be "debounced:" that means allowing a circuit or the IoT device itself to determine when the button has been fully pressed.

Proximity Sensors

There are several ways to evaluate the proximity of objects. These range from magnetic activated reed switches to regular limit switches. Other ways to determine whether an object or person is too close are antennas and sonar.

Computer Vision

Computer Vision is a separate topic. It is the reason why one may want a larger development board for one's IoT device; to run the open source package OpenCV, for example.

Environmental Sensors

We have already looked at a temperature and humidity sensor. There are other sensors for barometric pressure, air speed, radiation and so on.

There are far more sensors and devices for the Internet of Things than we can outline in this book.

Since space is limited here we must leave an inventory of the variety of parts that can be used to create Internet of Things devices to a more complete analysis. We have spoken about the Microprocessor (MPU, CPU) or Microcontroller Unit (MCU) but we have left out material about the sensors that will be used with our projects. We have not covered the variety of communications strategies that we might use (Cellular, Wireless Internet, Bluetooth, LoRaWAN) that can enable our device to report back to the Internet or Extranet. Finally we have not covered displays. It is nice for an embedded system to have some indicators so that an operator in-the-field can be enabled too!

Schematics

In the chapter on Integrated Development Environments (IDEs) we introduced Fritzing, a program that helps one design breadboards. It also produces schematic diagrams. Schematic diagram are very useful to show how the different electronic parts should be wired.

Reference Design

When we complete a circuit and create a wiring diagram or schematic it becomes what we call a reference design.

Displays

It is sometimes nice to have an indicator panel on your Internet of Things devices. These devices may be almost entirely "stand-alone" and require some operator input. It's a lot easier to touch a touch screen than log in via secure shell over TCP/IP or to use a serial upload link.

Perhaps the simplest sort of output for a digital computer is the Light Emitting Diode (LED). They are used for single lights and digital readouts.

These are nice displays that display a lot of data in a small space without using a lot of power. The OLED displays are an example of these. Finally, Liquid Crystal Displays or LCDs are useful to show large amounts of text.

Wiring and Testing a TFT LCD Display

Getting a large flat liquid crystal TFT (thin-film transistor) display (LCD) to work properly was difficult for me. Wiring a simple circuit on one breadboard with the help of Fritzing sketches may seem easy enough but a more complex device like an LCD Display with multiple data, power and timing requirements can become a daunting project. It soon becomes obvious that every connection must be secure, and tested. It took me several tries to get a 4.3" 480 by 272 pixel display working and I made several mistakes.

I will note that good diagnostics really helped me to "align" the software and hardware required for successful operation of the display. Without a good oscilloscope I might not have gotten the display operating at all. Even using the oscilloscope, I had some questions about the scale in microseconds by which the display was operating.

Equally important as good diagnostics to describe how a circuit is operating are the technical notes and documents for the products and devices being used.

Using Bluetooth in your projects 8

An exciting protocol for wireless serial communications is Bluetooth. This is the first time that we have mentioned wireless comminications in these chapters but we intend to focus on Bluetooth at the moment. There is often a need to transmit data from a device to a larger system, or as we saw in the case of the Arduino Thermometer, to the Internet. Serial and Parallel wired communications only go so far. We really need to have a networking protocol running on wired ethernet or over Wifi to reach the Internet.

The Arduino, especially the Arduino UNO used be small projects really can not send TCP/IP (Ethernet) out so readily. To do so requires a larger processor such as the Raspberry Pi. However, having found the Arduino to be so accessible for building computers that can handle the data and sensors required for the Internet of Things, it is worth our time to get Bluetooth up-and-running. This will allow us to use standard programs and utilities even from our cellphone to interact with our devices.

The Blynk dashboard that is available at www.blynk.cc is an example of a Internet dashboard that can accept IoT data from a device as small as an Arduino project. Our first step is to connect our project (the thermometer) to a Blynk server running on our ell phone via a Bluetooth adapter. One example of this is the HC-05 Bluetooth module. Using this device, data will be sent over the air (2.45 MHz) using the same standard serial transmit and receive connections that the ATMEL 328P processor uses for direct communications. Of course, setting all of this up can be a challenge.

Bluetooth Blink

In our next Blink example we will use the HC-05 bluetooth module to remotely blink the light emitting diode from a serial terminal on our smart phone. We have wanted to take the next step toward building smart things for the Internet of Things. Finally we are there!

The Arduino ATMEL 328P usually communicates with our Integrated Development Enviroment using the Serial Port however in this example we use the Bluetooth module to do so. Now when we send a '1' from a serial terminal on our phone to the Arduino sketch via Bluetooth the onboard LED goes on!

```
#define ledPin 13
 int state = 0;
 void setup() {
      pinMode(ledPin, OUTPUT);
      digitalWrite(ledPin, LOW);
      Serial.begin(9600); // Default communica-
tion rate of the Bluetooth module
}

 void loop() {
      if(Serial.available() > 0){ // Checks
whether data is comming from the serial port
        state = Serial.read(); // Reads the data
from the serial port
     }
     if (state == '0') {
      digitalWrite(ledPin, LOW); // Turn LED OFF
      Serial.println("LED: OFF"); // Send back,
to the phone, the String "LED: ON"
      state = 0;
     }
     else if (state == '1') {
      digitalWrite(ledPin, HIGH);
      Serial.println("LED: ON");;
      state = 0;
     }
 }
```

We need to use a serial terminal on our cell phone to send text strings to this Arduino circuit.

You may wonder how we will continue to communicate with the Arduino UNO, indeed program it, if we are using the one onboard Serial Port. The answer is that we can either temporarily remove the Bluetooth module while we program the Arduino UNO or we can use a Library (you may want to read about what are software libraries) called DebugSerial which will give us a second Serial connection.

A Bluetooth Control Panel: Blynk

It is a final step for use to use the Blink program we have written for a Control Panel on our cell phone. An IoT Dashboard like Blynk at blynk.cc can help us along our way. Using Blynk we register an Authentication Token which we use in an Arduino sketch (shown). Then over Bluetooth we can read the temperature from an Arduino Thermometer on a control panel on our phone (Android or iPhone).

```
/*************************************************
*************
  Download latest Blynk library here:
    https://github.com/blynkkk/blynk-library/
releases/latest

  Blynk is a platform with iOS and Android apps
to control
  Arduino, Raspberry Pi and the likes over the
Internet.
  You can easily build graphic interfaces for
all your
  projects by simply dragging and dropping
widgets.

    Downloads, docs, tutorials: http://www.
blynk.cc
    Sketch generator:           http://examples.
blynk.cc
    Blynk community:            http://communi-
ty.blynk.cc
```

```
    Follow us:                  http://www.
fb.com/blynkapp
                                http://twitter.
com/blynk_app

  Blynk library is licensed under MIT license
  This example code is in public domain.

 *************************************************
**************
  Note: This only works on Android!
        iOS does not support Bluetooth 2.0 Seri-
al Port Profile
        You may need to pair the module with
your smartphone
        via Bluetooth settings. Default pairing
password is 1234

  Feel free to apply it to any other example.
It's simple!

  NOTE: Bluetooth support is in beta!

  This example shows how value can be pushed
from Arduino to
  the Blynk App.

  NOTE:
  BlynkTimer provides SimpleTimer functionality:
    http://playground.arduino.cc/Code/SimpleTim-
er

  App project setup:
    Value Display widget attached to Virtual Pin
V5
 *************************************************
**************/

/* Comment this out to disable prints and save
space */
#define BLYNK_PRINT Serial
```

```
#include <BlynkSimpleSerialBLE.h>
#include <SoftwareSerial.h>

//SoftwareSerial SwSerial(0, 1); // RX, TX



// You should get Auth Token in the Blynk App.
// Go to the Project Settings (nut icon).
char auth[] = "<Get Your Own Token>";

SoftwareSerial SerialBLE(10, 11); // RX, TX
const int sensorPin = A0;
float voltage, temperature;
int sensorValue;

BlynkTimer timer;

// This function sends Arduino's up time every
second to Virtual Pin (5).
// In the app, Widget's reading frequency should
be set to PUSH. This means
// that you define how often to send data to
Blynk App.
void myTimerEvent()
{
  // You can send any value at any time.
  // Please don't send more that 10 values per
second.
  Serial.println(temperature);
  Blynk.virtualWrite(V5,temperature);
}

void setup()
{
  // Debug console
  Serial.begin(9600);

  SerialBLE.begin(9600);
```

```
  Blynk.begin(SerialBLE, auth);

  Serial.println("Waiting for connections...");

  // Setup a function to be called every second
  timer.setInterval(1000L, myTimerEvent);
}

void loop()
{
  sensorValue = analogRead(sensorPin);
  voltage = ((float)sensorValue / 1024) * 5;
  temperature = (voltage - .5) * 100;
  Blynk.run();
  timer.run(); // Initiates BlynkTimer
}
```

The Blynk dashboard was our first venture into the Internet of Things. Seeing temperature and humidity values on our smart phone was a first step in reporting data, though not directly over the public Internet. The project still falls under the moniker of Internet of Things however: a first step on the way for communications regarding electronic devices, things, to occur between computers, and, humans.

9 IoT Operating Systems

Don't forget about the very popular Raspberry Pi small board computers based on ARM technology and all of the related boards. For a larger IoT project these boards may preferable. Wiring small projects with the Arduino Uno is a great first step in understanding the sensor-side of the Internet of Things, and also learning basic protocols for communications. A larger project may require a board capable of running an embedded operating system.

It may seem logical, having found the Arduino toolkit, to keep expanding using some of the larger Arduino boards such as the Arduino Mega that are available. And to be fair this is a reasonable idea. At some point having a computer system capable of running standard programs on a Linux or Windows operating system may be preferable for an Internet of Things project than a dedicated microcontroller. Using an operating system we can run small Internet servers that speak the language of the Internet; that is provide information to Internet consumers. Or we can follow the instructions on one or more commonly available Internet Dashboards to send sensor data to a control panel over the Internet. Whatever our reason for using the Internet of Things, be it reporting data, managing Inventory or remote control, a computer that can handle an operating system can make a big difference.

In our previous example, the IoT Thermometer based on Arduino requried a lot of software to be written. Are we going to throw all of this work away? Rather than dropping Arduino now that we have had such success with it we may wish to incorporate it into

our Internet of Things projects. Like a programming language, the Arduino pinout has become a standard. Now imagine that instead of just monitoring the temperature and humidity for a building we wished to report weather conditions. We might rewire our device using the input and output pins of a different processor. Or we could instead use our Arduino device to send data back to an Internet of Things server. Finally if we had developed a shield for the Arduino we might use this on a different device as well.

Without needing to complete arduous throughput calculations it is clear that the Atmel 328P or the Arduino UNO is not quite powerful enough to monitor several sensors, cache the weather data for a substantial length of time and transmit the data. Of course, in choosing the popular ARM-based Raspberry Pi we now have to learn a whole new toolkit.

Before you exclaim, I don't have a weather station, we remark that we don't have on either. What can be done? At the other side of the basement we can start printing out plastic eggshells to measure wind speed. Or we can repurpose plastic waters to measure rainfall. When we are done we have an ARM device that we can use to send weather data over the Internet, possibly using an IoT control panel.

10

Intel x86 Boards

This book originated as guide to x86 based embedded processor boards. What has happened to these boards? Actually, Intel Corporation, which introduced the Intel Galileo and Intel Edison back in 2015 still makes a development kit based upon the Intel Quark D2000 Microcontroller Unit. These boards run Arduino compatible shields, though not as fast as an actual Arduino will. But running x86 is very important for compatibility with a lot of good software such as Microsoft Windows and Linux. As we will discuss not all programs can be "ported" to ARM technology even if the ARM chips are better for running Internet of Things applications. And certainly a small embedded processor like the Arduino cannot run programs that depend upon at least an embedded operating system.

UDOO, UP and Latte Panda

Although the author has only tried the UDOO board, an x86 board with a second Intel x86 Arduino 101 processor on board, there are a number of boards on the market that run or are compatible with Intel x86 software. We add this further caveat, compatible, because AMD, Intel's rival in desktop processors has always had x86 compatible systems based on their own opcode set and architecture. Also NVIDIA Tegra looks like a good entry in the robust IoT market

An Inventory Control System

An Inventory Control System might be a good example of a reason to

use Intel x86 IoT. Imagine that you have a business which sells 'widgets.' that is, a small electroncis sotre that sells electronics components. This is a basic Use Case (see Glossary) if we ever heard of one! But wait a second! Our packages must have barcodes or QR codes so that we can keep track of everything!

It will require a full operating system to maintain more complicated and more aware Internet of Things computers like an Inventory Control System. This would be a system that can count the parts in every drawer, or organize them using some mechanical sorter.

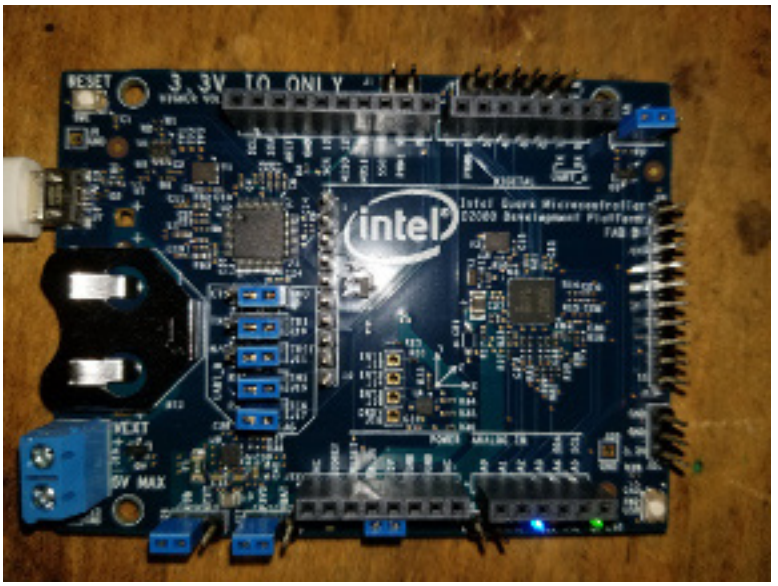
To do this we break down the activities of the device into functional units. It might require the use of a Venn diagram to decide what subsystems would be implied by these uses. It is best to start with a clear statement of what we hope to accomplish. Take for example our need to sort and keep small electronics parts organized. At first it might appear that some sort of "Robotic Squirrel" could deal with the parts drawers and piles of loose parts on our desk. But when we think about how we really might accomplish our goals of a clean workshop, using some automation, we might retrofit a model railroad instead. Now in addition to a clean workbench we will have an article for hobbyists entitled "How I built a 'Working' Model Railroad!" Choosing the archetypal design prototype of a railroad simplifies our use cases quite a bit.

Imagine the parts that would be required to create this Internet of Things railroad. We will require a lot of small parts to make this work. In addition to the parts required by this device it must also identify the many spare parts that it is required to sort. To do this it may need computer vision coupled with online Artificial Intelligence sufficient to find the specifications of the various parts.

If we were to create a database of the Things of the Internet of Things we would have to add a section for all of these devices! But before you exclaim that this example is contrived or perhaps not funny try to imagine both what would be required for you to build it yourself, and how a real-world application would be able to better operate using such a device.
A better idea than a database of the Internet of Things that has no end is a comparison between the functionality of an Internet of

Things device such as the thermometer above and an actual thermostat. We would need to list the features and abilities of both units. The same would go for the other devices. How would we expect an Internet of Things parts sorter to behave? How would this behavior differ from the behavior of an ordinary, operator-centric, non-IoT device?

After we compile this list, we will be in a better position to discuss the commonalities between Internet of Things devices and start to create block diagrams of their operation.



An IoT Frisbee 11

Creating software is key to building Internet of Things devices. Since the first two projects used the basic Arduino controller to operate devices we will explore the Neural Network of the Arduino 101 to build a Frisbee that will report back to viewers on the Internet as to its progress. This will be a way of testing the new model of Internet of Things and comparing our Frisbee designs to others. While we are improving our Internet of Things models are we also becoming aware of the ability of neural networks to make decisions about computations on the very edge of the network?

What are the building blocks required to build this Frisbee? There are several. An onboard Arduino 101 chip may tell us the tilt of the Frisbee as it flies. An onboard wireless internet board chip might relay communications back to a central hub computer. Finally, there could be a (very) small touchscreen on the Frisbee to program its electronics.

At the Edge Computer, the hub, the work is only beginning. If the Frisbee is working correctly, it will be sending out a steady stream of data regarding its flight: but only when it is thrown. The edge computer sitting at the base camp for our Frisbee game has the responsibility of tabulating this data, and, if it is truly to be managing a Frisbee that is on the Internet of Things, to offer this information to other computers as a service to people on the Internet. Now everyone playing Internet Frisbee can share their data.

As referenced earlier, these days we are using 3D printers to build the things used by the Internet of Things. We do not want to buy

every component required to bring the Internet to a Frisbee only to find out that it is too heavy to fly. So how might we design and build this novelty?

We must prototype the circuit described anyway. We can go ahead and assemble the electronics and test its computing functionality in a laboratory while we figure out the problem of aerodynamics.

Getting Started

The devices we hope to build are built from discrete components but more important in the design process are the functions we hope that they will perform. Though the parts will be as distinct as individual parts such as computer chips and electronic parts to entire subsystems that we use together, a good use case will include all of them. Say, for example, that we wished to create an Internet of Things Frisbee. A block diagram of this Frisbee would include sensors and communications devices as well as a microprocessor and battery power. The electronics would have to be light enough to fit on an aerodynamically competitive Frisbee. But the block diagram of the use cases for this Frisbee will outline how we will expect the Frisbee to perform; for example, when it is thrown it may transmit data back to the base. Depending upon what we hope to gain from building this device we will have different functions that it will perform. Then we can go on to decide upon what devices to use during its construction.

Starting the Design

We have to find light parts that can be aerodynamically built into the frisbee.

12 Software Tools

They say that one works best if one enjoys ones work. Right now I am compiling Cross-Compilers for the TI MSP 430 and other ARM chips.

You have spent a lot of time on your Internet of Things project! Be preprepared to spend just as much time getting it to work.

I had to look up the definition of "Operating System." My best definition is a way that a peron can tell a computer what it should be doing. Although getting small board computers to operate correctly can be as hard as the full-size computers, don't let that scare you. We will start out easy by downloading images to flash memory drives to place on the computer board. In some ways these memory cards are not as good as the old Programmable Read Only Memory (PROM) chips that used to hold the operating system for small computers before because they are so flimsy, and the latches used to hold them are not that secure. Nonetheless we will cover three ways of getting an image onto the board: downloading the software, building a fresh kernel, building stand-alone programs. Depending upon the chipset you are using one or more of these techniques may be suitable.

I don't have any real need to get these chips and compilers working but they will make a good entry in the book. I need a whole section explaining how I built and used the ARM toolkit, just as I need a section describing the YOCTO build environment. It took a long while to figure these out and someone could benefit from the discussion.

Actually the TI chips might be good for controlling the Inventory Control Unit. The Chips are cheap and you can use a lot of them. I have to explain why one would want to use so many CPUs in the book and consider the Budget for IoT; including the power consumption budget.

Also the ARM compiler would be good for the NXP chips. I have forgotten about them. These are useful chips for projects. You need the ARM compiler for them.

The third path that I need to cover is merely downloading the OS to a memory card. That can be complicated enough. This is actually the first of these there chapters

Without software the embedded computers are useless. Yet there are a lot of steps required to create a software system for an embedded device. This process often includes building a software release and downloading or "burning" it on the small board computer. Alternatively one may use off the shelf software and merely reconfigure it: this may also require some effort as operating systems have to be chosen and downloaded or libraries for a specific application installed.

Security, Backup and Recovery

Very important! The security of Internet of Things devices is shaky at best! And, as with all computer systems there should be a backup and recovery solution in place for a critical device. A good place to start for Internet of Things security is Norton and Beyond Trust.

The Importance of Documentation

And a unified document store. Of course, that is what the Internet itself is supposed to be. But it might be important to use unified resource identifiers within one's own knowledge base. Wait a second! What is a Knowledge Base? This book itself is a means for documentation.

Development Tools and Software

All of these tools and toolkits are designed to enable our devices to run software. As we mentioned early in this book there are a lot of people and a lot of different applications involved with the Internet of Things. At times it has seemed as if we have been writing more about computers than devices but the point is that 'things' have become inseparable from our thinking about and working with them. Sooner or later however the software must work to actually accomplish something. How do we do this?

In short we can say that our need for working software is the reason for learning Object-Oriented design and development and learning skills such as building your own Linux Driver.

Many standard operating systems such as Linux or Windows CE also come with embedded software development tools.

Yocto

Where would we be without the resources of the Interet! With just a Google search or a query on Stack Exchange or a myriad of other sites we can find information about the technology with which we are concerned. And what could be more elemental than rebuilding the operating system by which many computers run.

A lot of people don't know what is required to reload the operating system or the "brains" of a computer, much less know what an operating system is. Though there are different definitions the one I like best is "a computer program by which a human being can tell a computer what programs to execute and when they should run."

The wonderful part about open source technology is that the source code -- that means uncompiled computer programs - of the linux operating system are freely available. Sometimes there are reasons to rebuild the kernel rather than depending upon the "build" that someone else has provided. I like to obtain linus from kernels.org and recompile it on my Slackware install.

The build environment really does matter. First you have to know what kind of computer you are using to build the new kernel. Even an old Apple PowerPC can be reprogrammed with Linux and the new kernels for Raspberry Pi and other boards can be developed there. This is called cross-compilation. In the simpler case you simply rebuild and update the existing operating system, performing all of the development on the computer that will be the target machine

A good system for building an embedded operating system is Yocto at yocto.org

13 Internet Dashboards

The edge computers upon which we have so laboriously toiled must be visible to "dashboards" available on the world wide web, or have some other iteraction with the Internet to truly be called Internet of Things devices. But to reiterate an earlier point, a smart or useful device that is not online may be all that we need.

Although the implementation of an Internet of Things dashboard is beyond the scope of this book we should describe some next steps that a designer might take to be ready for these systems. Ultimately an Internet of Things device that provides a service over WiFi or Wired Ethernet, LoRaWAN or other IoT Telephony such as 5G could be used by a web dashboard, even one that we had built ourselves. We mention this last, because there are a number of consumer-oriented Internet Dashboards out there, like the Blynk dashboard seen earlier, that are ready-to-go.

Web Services

When a web page receives data a browser actually reads HTML from a web server. In the same way, a web server that needs information such as the current temperature from an Internet of Things device may obtain this data from a web service using a protocol such as TCP/IP, the transport mechanism of the Internet itself. The Internet of Things device, or web service is said to "publish" the data to the Internet, or in the latter case, to the web server, which "subscribes" to the data stream from the IoT device. The dashboard is

then able to construct a webpage that is visible on the public Internet.

14

Conclusion

Throughout this book we have learned to use the computers and protocols of the Internet of Things to create real-world devices. As we mentioned in the Introduction, the Internet of Things is a paradigm shift in which the devices and items of our world themselves use the Internet. But we don't have to be helpless in the face of this new world. We can build Internet of Things devices ourselves. We conclude this book by reviewing the steps and technologies needed to build an Internet of Things device.

Sensors were critical to our implementation of Internet of Things devices. This is the first place that an edge computer acquires data from the world. Although we only used a temperature and humidity sensor for our project, we learned how this data is obtained by a small embedded computer (Arduino Uno). We discussed the myriad of other available sensors and migrated our project to a larger computer (Raspberry Pi) in order to support a whole weather station. At least we started that project. As we mentioned we might need to print out some parts on our 3D printer to complete this device.

In focusing on the Arduino thermometer, we were able to send data to an Internet dashboard using a bluetooth connection in much the same way that a larger IT project would operate. We also discussed the use of the Internet of Things for other real-world applications such as inventory management. There is a lot more to the Internet of Things than just the dashboard.

Who uses all of the data created by the Internet of Things and what

do they use it for? That is a good question that is beyond the scope of this book. However our final design, an Internet of Things Frisbee showed us that we can model the use of data on the Internet in much the same way as a team would play a competition online.

Also, out of the scope of this book were neural networks and machine learning systems that can be included in Internet of Things devices. For although we reported data about environmental conditions, we did not consume Internet of Things data to allow our devices to make informed decisions. But this is part of the Internet of Things as well. All-in-all we were fairly granular in describing how one or two edge computers on the Internet of Things might operate.

We opened a lot of doors in this book and covered a lot of material in an attempt to master the design and functionality of end-point Internet of Things devices. Now it is your turn to build your own machines and use them in your environment.

Have a creative journey!

Glossary A

3-Tier
ACGNJ - Amateur Computer Group of New Jersey
APU
Back-End
CPU - Central Processing Unit
Distributed Computing
Front-End
Full Stack
GPU
Internet of Things
MCU - Microcontroller Unit
MPU - Microprocessor Unit
Operating System
TCF - Trenton Computer Festival
TCNJ - The College of New Jersey
Use Case – A way in which a device is supposed to be used.

B

Safety with Electronics

So far we are doing great with our devices and electronics. A word on safety precautions that we shoul take to protect these devices and ourselves is in order.

When we speak of safety procedures to use with electronics we are not referring to electrical safety, or are we? Don't forget that although they use very low voltages (1.8v to 5v) and low current in each wire that the electronic devices in use within a computer are electrical in nature.

The same rules regarding prcautions to take regarding electric shock apply to all electronic equipment.

I am not the authority on these rules; suffice it to say that there can be high voltages within a computer enclosure that can hurt you.

Also the electronic cirucits are themselves susceptible to electronic shock.

1) Be sure to use the right input power levels.
2) Using a wrist grounding strap when handling sensitive electronics.
3) Observe the proper polarity of electronics in a circuit.

Epilogue

One thing I like about writing is that one can write about real or imaginary things without actually being there. And what is the future Internet of Things but things imagined? Although there have been practical, hands-on technical examples in this book, the designs and considerations involved theirein are intended to be equally instructive and explanatory. I hope that I have helped you to take next steps in your Internet of Things projects.

Evan Jan Williams

Text is set in Gentium, 12pt.
Printed by Bookmasters, Inc., Ashland, OH

Building Computer Systems for the Internet of Things

When one first hears the term "Internet of Things," one may become a bit frightened. Are machines really taking over our world? In Building Computer Systems for the Internet of Things, Evan Jan Williams takes a technologist's perspective on this situation by asking "What is required to build the 'edge computers' that power the Internet of Things. It is his hope that we will be better able to build and use these devices ourselves.