Working Electrical Engineering Projects with STM32 + Fun Kit

Daniel Beatty

April 21, 2021

Abstract

Abstract

1 Social Media Overhead

For all of the videos produced for this series, would you utilize the like, subscribe, and notification features of this streaming platform. YouTube like any social media utilizes these pieces of data to drive the statistical algorithms of their advertising machine.

This channel also plans to initiate a patron page for this content.

Any and all support from you, my faithful audience, contributes greatfully to my success. For that and your participation, I am truly appreciative. Thank you.

2 Introduction

Welcome to this course on developing embedded software on Advanced RISC Machine (ARM) Limited family of platforms. This course is part of a series ‘‘Back to the Basics - an Electrical and Computer Engineer’s perspective on software development.” This series also examines hardware development.

As I journey to pick up my professional engineering license in California, I am humbly reminded that there is no such thing as a PE for Software Engineering in California. The grandfather class that protected me in Texas doesn’t exist here in California. Therefore, all I have is the grandfather clause from Texas and our professional societies. The Institute for Electrical and Electronic Engineers (IEEE) and Association for Computing Machines (ACM) have done much to support us as professionals in all aspects Computer Science and the engineering disciplines that follow.

The work developed for this course focuses on the needs of my clients. This job needed a micro-controller to manage a device to drive large power loads. Therefore, this document focuses on microcontroller features. These features form the backbone of the internet of things.

This course documents a journey. The work I have done to refresh my understanding of embedded systems takes a cross platform approach. I have picked interchangable general purpose platforms (GPP) used to develop software. Whether I utilize Apple’s, Microsoft, or Linux as my GPP and tool vendors is a matter of personal preference to fit my needs. Customers, students, and collegues may have different preferences and needs and that is alright.

Bottom line, I want to deliver a good products to my clients. The more effective and efficient I am helps out substantially. Even I have to refresh my knowledge base.

So what makes for a good product? The architectures selected tell an engineer about the product’s goodness score. Object oriented design stands on the pillars of design patterns. A computer engineer applies algorithms, design patterns, and analysis engines. These things give the engineer confidence.

To do this, this work focuses on the mBed system of development. This document shows this project utilizing the STM32 product line. In some cases, I also utilize the Arduino products just to demonstrate a cross manufacture approach to testing designs. Lastly, this document shows how to do this work on the MacOS, Windows, and Linux (ARM) platform. Each of the product used in this course shall be provided in the lesson description.

3 The Base Line Integrated Development Environment Part 1

The Eclipse Foundation and Advanced RISC Machines Limited (ARM) together produced mBed Studio. The mBed Studio provides an Integrated Development Environment (IDE)[1] . ARM designed their IDE for micro-controller developers to simplify complex projects.

ARM included a hardware abstract layer (HAL) to connect common features from many producers. Thus, I can develop a simple program, compile it for each board, and that program will run. ARM connects these HALs together by communication methods.

Mbed Studios (a modified Eclipse Theia) released in June 2019. It is derived from the Eclipse 'Theia' line shown in a YouTube video[2].

ARM supplies a tutorial on installing the mBed Theia environment on MS Windows 10[3]. I am doing the same on the MacOS platform.

This background topic shall show examine basic mBed tasks on Intel based processor machines running MacOS and Windows. This work also demonstrates the same tasks on Raspberry Pi.

3.1 ST Microelectronics Libraries

ST Microelectronics (STM) started as a French company. Now, STM operates out of many countries. In addition to the boards themselves, they also produce support libraries for their boards[4].

For example, STM and Texas Instruments (TI) produce ARM based boards with digital signal processor (DSP)[5] support. A DSP provides acceleration to vector math. These architectures enhance a MCU's ability to quickly match patterns and make adjustments.

[1] https://os.mbed.com

[2] https://www.youtube.com/watch?v=HsTtzqL-GP8

[3] https://os.mbed.com/teams/ST-Americas-mbed-Team/wiki/Getting-Started-with-mbed-and-the-STM32F

[4] http://www.emcu.it/STM32F4xx/STM32F4-Library/STM32F4-Library.html

[5] https://www.st.com/content/st_com/en/products/embedded-software/mcu-mpu-embedded-software/stm32-embedded-software/stm32-standard-peripheral-libraries/stsw-stm32065.html

3.2 Hardware Abstraction Layer(s)

STM, TI, and Arduino all benefit from mBedOS' Hardware Abstraction Layer(s)(HAL) to access key MCU capabilites. The mbedOS is not the only real time operating system (RTOS) for micro-controllers. ARM's mbedOS does support many implementations of its chip set [6] design. The mBed compilers build both the mBedOS and specific project code into a tidy size. This good support makes ARM chipsets a safe bet for many manufacturers.

The mBed website carries white paper references for each of the boards it supports. For example, this document utilizes the Nucleo-F413ZH[7] as an example. This platform and others can be found at [8]. Alternative cards include:

- Discovery F413H [9]
- Nucleo L4R5ZI-P [10]
- Nucleo WB55RG [11]

Some of the main library sources from STM are provided on the STM32f4 Discovery web site[12] forum.

[6] https://os.mbed.com/questions/53876/CMSIS-vs-STM32CUBEHAL-vs-MBED/
[7] https://os.mbed.com/platforms/ST-Nucleo-F413ZH/
[8] https://os.mbed.com/platforms/
[9] https://os.mbed.com/platforms/ST-Discovery-F413H/
[10] https://os.mbed.com/platforms/ST-Nucleo-L4R5ZI-P/
[11] https://os.mbed.com/platforms/ST-Nucleo-WB55RG/
[12] http://stm32f4-discovery.net/2014/05/all-stm32f429-libraries-at-one-place/

3.3 Your Workspace

Stephen C. Johnson of AT&T's Bell Labs developed a portable compiler and released it in 1979. The initial committee was spearheaded by Alan Snyder in 1973[13]. Borland made strides to port a Pascal compiler in 1983 and later a C/C++ compiler. The notion of a workspace emerged as common practice both on PC and on Unix platforms by about 1991.

Eclipse quickly became the IDE of choice for Java by 2003. Contributors to the Eclipse platform, notably IBM, provided multiple platform support. The Foundation provided simple recipes that software developers could to get up and producing for free. In 2003, new Service Oriented Computing Environment (SORCER) laboratory sold the Information Technology (IT) Committee on the need. The IT team quickly deployed Eclipse in the labs. Eclipse then ran Mac, Windows, Linux, and Solaris laboratories.

A workspace is simply a directory that an IDE utilizes to manage its projects. Companies and universities deployed the Concurrent Version System (CVS) with their Unix systems as early as 1986.

CollabNet created Subversion (SVN) in 2000 with Unix and Windows. CollabNet supplied SVN as an open source Version Control Systems (VCS) starting in 2009.

Mbed Studio inherits these integrated features from its Eclipse Theia roots. Mbed Studio detects the artifacts from these VCS to determine which the developer uses. Therefore, Mbed Studio provides tools to interface with the VCS and facilitate management thereof. The Agile development approach found in web application shops utilizes VCS, dedicated build systems, and code search browsers to enable team development. These systems track code to include the working state builds. Does the code compile? Does the code pass unit tests? For code trees separated from the main branch, what new features does it have? These issue allow the team to determine when to integrate new features in their product.

For the internet of things, developers can integrate these capabilities with system on a chip (SOC) systems. These systems help to manage embedded systems updates. This capability provides users and technicians a mechanism to manage their collection of systems.

All of these capabilities stem from the basic notion of a workspace. The workspace is the developers first tool of managing the software they develop on any level. Build your workspace with care.

From the workspace, you can create programs from templates. The first examples presented in this text only utilize blank and blinky template programs. Most of the others require the utilization of network establishing components. Some boards and devices build these features in and others don't. The more interconnected your product is entails the need for further cyber security to protect it from abuse.

In another chapter, this text shall introduce the notion of a team / distributed build system approach. What makes such system different is the level

[13] https://dl.acm.org/doi/10.1145/512760.512771

of detail required to ensure that a produce developed by one developer integrates with the products of others. Lastly, such systems should also enable distributed products to upgrade their software with confidence.

3.3.1 Creating A Program

The mBed Suite documentation provides an excellent how-to guide on creating a program [14]. This process works out in a very straight forward way.

3.3.2 Importing A Program

This process is a simple file system operation combined with applications for your favorite version control system. In this example, this work presents GitHub. For the reader's purposes, any version control system can work. Teaching these version control systems is outside the scope of this work.

3.3.3 Create a Repository

We present software creation from a independent producers point of view. Many corporations and government entities insist on their own repository tools. For these entities, we can comprehend the need of such restrictions. They have assests to protect and the finances to provide on property protection.

On the other hand, your work is often times things that you want the priviledge to reuse. Reuse is one of the tenets of object-oriented programming. When we restrict one of these tenets, we risk detriment to our profession.

So, we go through a typical arrangement with to produce a repository for use in embedded development. Providers can vary such as BeanStalk and GitHub. Resources such as GitHub come in handy as mBed has much of their resources available through this resource. However, ARM can just as easily backup their repositories via other outlets, including the hosting of their own.

[14] https://os.mbed.com/docs/mbed-studio/current/create-import/index.html

3.3.4 The simple C++ class

The ANSI, NIST and ISO committees; who have taken an interest in C++ language; developed several iterations to capture the basic notion of a class. For many students of computer science, the C++ language is the first language learned. ANSI/ISO has governed the standard by which each generation of student has learned the notion of a class.

The basic notion of a class provides an encapsulation of abstract data types. Basic automata theory captures some of the basics of object oriented programming as input, states, and outputs.

So, we create a simple class to spell out Hello World.

```
#include "mbed.h"
#include <iostream>
#include <cstdio>
#include <string>

// main() runs in its own thread in the OS
int main()
{
    std::string shortCircuit = "Don't disassemble ";

    std::cout << "Try 1 " << std::endl;
    std::cout << shortCircuit << " = number " << std::to_string(5)
              << "." << std::endl;

        return 0;
}
```

In C++, there exists two notions of a string. Most 1980's and 1990's students of computer science remember the character array. In the 2000's the notion of a vector to dynamically adjust for growing data entered in the standard's body of attention. So, a string class entered the standard template library (STL) that exhibited traits similar to those found in Java and Objective-C. This approach does bear fruit for features useful in control devices.

A simple method of devising a C++ class includes the simplest program taught, namely "Hello World." Some also pluck silly little phrases from 1980's movies like 'Short Circuit'. To illustrate a little mastery of data types, we add to the class some simple integer and floating point members.

A few things to remember in developing a C++ class. One, we use the macro system. Compilers complain if it finds the same class defined twice. The include system needs the macros to ensure that a class is defined only once. The define needs to differ in some way from the name of the class or struct (say an all caps form of the name, where as the name itself just starts with a capital letter). Lastly, this example demonstrates a class with a primitive form of marshalling.

```
#include <iostream>
#include <cstdio>
#include <string>

#ifndef HELLO_WORLD
#define HELLO_WORLD

class HelloWorld
{
    std::string message;
    int units;
        float cost;
    public:
        HelloWorld();
        HelloWorld(int units = 0);
        HelloWorld(int units, std::string &message );
        HelloWorld(int units, std::string &message, float cost );
        std::string getMessage();
        int getUnits ();
                std::string marshal();

} ;

#endif

std::string HelloWorld::marshal()
{
    return (this->message + " = " + std::to_string(this->units) +
                                    " I cost " + to_string(cost) +".") ;
}
```

Thus, we develop the notion of concatination of strings. The STL string object has a to_string function that exists to make strings out of primatives. In later chapters, we shall utilize the string object to parse simple language statements into objects and vise versa. This method "marshal" simply shepards the text and simple data types as one simple sentence.

Then we simply have a new version of the program that sends the string to the C-output stream.

```
#include "mbed.h"
#include <iostream>
#include <cstdio>
#include <string>
#include "helloWorldStruct.hpp"

// main() runs in its own thread in the OS
int main()
{
    HelloWorld alpha(5);
    std::string shortCircuit = "Don't disassemble ";
    HelloWorld beta(5, shortCircuit);

    std::cout << "Try 1 " << std::endl;
    std::cout << alpha.marshal() << std::endl;

    std::cout << "Try 2 " << std::endl;
    std::cout << beta.marshal() << std::endl;


    std::cout << "Try 3 " << std::endl;
    std::cout << beta.getMessage() << " ."
              << beta.getUnits() << std::endl;

        return 0;
}
```

Here we use the first constructor for an object called alpha. We construct this object with merely the unit number. We construct the second object (beta) with a message. We then print these messages. Note, the direct notion of just grab the members fails to print the units. This is a stream issue in mbedOS.

3.4 It is about the Libraries, Shhsss.

For any task associated with a MCU, ARM based micro-controllers require the libraries. ARM MCUs require the frameworks of the RTOS itself, in this case mBedOS.

Developers refactor their code as they get into development. These libraries serve as building blocks. This product explores basic library creation to benefit fellow developers and makers of products.

As we create libraries and programs, it behoves us to consider license programs and how to market our productions. We want the fruits of our labor to bring us profit. If we never make and sell things then we certainly shall not build profit. If we horde what we make then we won't make profit, either.

So we explore creating programs, importing existing libraries, and creating new libraries. Documentation of such products contributes to our fellow makers. We make money from establishing those connections and making the software that connect to real life.

3.4.1 Importing Libraries

ARM made importing a program rather simple by utilizing the Eclipse Theia platform. The most common set of libraries import is the mBedOS itself. This framework contains both the real-time operating system (RTOS) mBedOS and its fundamental C/C++ libraries. It also contains libraries for other languages such as Python.

Demo:
[15]

3.4.2 Creating Libraries

So why did we just tell you about the C++ class. Simply put, any library we build should utilize namespaces appropriately. It should allow customers to use these libraries for their products, too. Yes, fellow developers may exist as a customer. This goes to the Object-Oriented Program mantra about sharing classes/objects.

This work demonstrates the creating a library first with the Hello World library example. Eventually, we introduce a real world example such as a Oscilloscope class.

4 Basic Hard Tools

4.1 Work on Timer

In cases where we ask the process to wait, we should ensure that the I/O pipe empties. Otherwise, we have no idea when this process takes place. It can occur

[15] https://os.mbed.com/docs/mbed-studio/current/manage-libraries/index.htmlimporting-a-library

in the wrong times.

There are third party USB drivers such as [16].

Also, the timer's have issue with floating point numbers.

Lastly, the STDIO has trouble with printing floats.

4.2 Flushing The Standard Input Output

Prior to mBedOS 5, the use of the C Standard Input Output library meant opening a USB serial port. Most of the C Standard Input Output library functions were methods of these singleton classes.

With mBedOS 6, the C Standard Input Output library uses the main USB serial port by default. Therefore, serial communication works the way a typical desktop/laptop environment does, with one exception. Threads and Interrupt Service Routines (especially timers) don't play nice with the buffer. The process of printing or scanning characters does not have the real time behavior when the either real time events occur. Thus, the developer does have to keep track to clear the buffer.

4.3 Blinky

For many in computer programming industries, the first program a developer writes is 'hello world.' The goal of such a program is to demonstrate that when developer runs the program that it does something the user can observe.

In this example, we develop a program called blinky. This programs main objective seeks to show some observable work. In this case, the developer can put together a external circuit or use on board lights of the Nucleo to show the program doing work.

Setup DigtialOut Class for pin PC_1
Setup DigtialOut Class for pin PC_0
while Always **true do**
  Toggle PC_1
  Wait
  Toggle PC_0
  Wait
end while

We can also do basic threads to allow the OS to handle wake up times. The basic algorithm for the thread control function is similar. The pin object needs to be relatively global to the name space. A good exercise is to build a class for blinky with pin and timer value for members. The actions of this class need to include its initiation with pin number id and default timer values. More actions entail starting and stopping the thread.

The ARM team changed the thread library to use the standard chrono space for its time measurements. For those of us who trained first on C/C++ 99 and earlier, use of this library seems foriegn. As shown in the listing, an object can

[16] https://os.mbed.com/users/gte1/code/STM32_USBDevice/

be created from this Standard Chrono namespace. mBed Studio also allows compound interpretation of number and unit as the same namespace object as shown in the main body.

```
#include "PinNames.h"
#include "mbed.h"
#include "rtos.h"
#include <chrono>
#include <ratio>

DigitalOut LEDA (PC_1);
DigitalOut LEDB(PC_0);
Thread thread1, thread2;

void LEDAControl()
{
    while(true){
        LEDA = !LEDA;
        std::chrono::seconds d(1);
        ThisThread::sleep_for(d);
    }
}



void LEDBControl()
{
    while(true){
        LEDB = !LEDB;
        std::chrono::milliseconds d(500);
        ThisThread::sleep_for(d);
    }
}

// main() runs in its own thread in the OS
int main()
{
    thread1.start(LEDAControl);
    thread2.start(LEDBControl);
    while (true) {
        ThisThread::sleep_for(10s);
    }
}
```

4.4 Seven Segment LED

The seven segment light emitting diode circuit example yields some interesting issues for a microcontroller. One, the basic digit requires one digital out pin per segment (in other words seven). Two, the circuit also requires a common ground. Three, if the circuit multiple digits involves multiple digits, these devices store logic levels. This allows another digital pin out set to determine which digit to change.

This issue is a classic issue dating back to early micro-controllers. This issue allows most instructors of micro-controller theory to segway a lesson from basic digital design into the curriculum. Therefore, we see a simple software definition of a seven segment LED structure to determine the digital output pins.

The mBedOS boards allow software to address whole port families. We can address ports A, B, C, etc. as a whole register. This makes the development of libraries and classes a much simpler exercise.

One word of advice, pick your use of ports carefully. If the controller needs to read in analog to digital, produce pulse width modulation, or digital to analog out then these operations can't share the port devoted to managing the seven segment LED.

The basic digital design portion of this exercise requires one to map the each digit to what pins must be turned on. This map must then be applied to an 8-bit word that controls the pins. Note, the designer can do this with individual digital out operations. It simply requires on the order of 8 sets operations more to do so by individual digital out instructions. Where as, assigning a single 8-bit word requires fewer instructions.

In this example, we use 5161AS seven segment LED provided in the Rex Qualis Uno R3 Project Starter Kit. There are other seven segment LED models made, and one must identify the segments to build the circuit properly.

The pins for the Nucleo-F413ZH can be found online[17]. The pins for a basic bread board with the 5161AS seven segment LED can be found in figure **??**. The port settings to equal the number to be displayed are found in table **??**.

Table 1: Seven Segment Display Port Values

Number	**PC 7**	**PC 6**	**PC 5**	**PC 4**	**PC 3**	**PC 2**	**PC 1**	**PC 0**	**Hex Value**
0	0	0	1	1	1	1	1	1	0x3F
1	0	0	0	0	0	1	1	0	0x06
2	0	1	0	1	1	0	1	0	0x5b
3	0	1	0	0	1	1	1	1	0x4F
4	0	1	1	0	1	1	1	0	0x66
5	0	1	1	0	1	1	0	1	0x6D
6	0	1	1	1	r5c6	1	0	1	0x7D
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7f
9	0	1	1	0	1	1	1	1	0x6f

[17] https://os.mbed.com/platforms/ST-Nucleo-F413ZH/

```
  Configure Port C Lower Byte as a group
  Define segment patterns
  Set Count to 0
  while Always true do
    Send count to display
    Wait 1 second
    if Count is greater or equal to 10 then
      set count to 0
    end if
  end while
```

```
/*
        This 7-Segment Counter is based on a textbook
example from Dogan Ibrahim's ARM-based Microcontroller Projects Using mbed.

        Modifications were made by Dan Beatty to adjust to mBed Studio and mBedO
*/


#include "mbed.h"
#include "rtos.h"

PortOut Segments(PortC, 0xFF);
int LEDS[] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f};

// main() runs in its own thread in the OS
int main()
{
        int count = 0;

    while (true) {
                Segments = LEDS[count];
                wait_us(1000000);
                count++;
                if (count >= 10) count = 0;
        printf("%d \n", count);

    }
}
```

4.5 Basic Digital Input

Basic digital input can be demonstrated by an 3 sets of switch input. The input goes low when the switch is closed and high when the switch is open. The switches are simple buttons provided in the Rex Qualis Electronic Component Fun Kit. This kit also has an RGB LED. The top of the LED is transparent and

one can see the etching. The fat etching is the common ground. We connect the other three to red, blue, and green control pins. These pins need a 330/390 ohm resistor to establish sufficient current limits for the Nucleo Board.

```
Configure the RGB output
Configure the R, G, B buttons as inputs and enable them as Pull-ups
Clear all LEDs
while Forever TRUE do
  if R is pressed then
    Toggle Red color for display
  end if
  if G is pressed then
    Toggle Green color for display
  end if
  if B is pressed then
    Toggle Blue color for display
  end if
end while
```

```
/*****************************************
In this project, an RGB LED is connect. So are 3 buttons.
Pressing a button toggles the corresponding color.

R button = PC_0
G button = PC_1
B button = PC_2

The RGB LED pin are connected as follows:
Red = PC_3
Green = PC_4
Blue = PC_5

Original Supplied by Dogan Ibrahim, August 2018 in
"ARM-based Microcontroller Projects Using mBed"
Modified by Dan Beatty for use in mBed Studio
mBed OS 6.x

*****************************************/


#include "mbed.h"

BusOut RGB(PC_3, PC_4, PC_5);
DigitalIn R(PC_0, PullUp);
DigitalIn G(PC_1, PullUp);
DigitalIn B(PC_2, PullUp);
```

```
// main() runs in its own thread in the OS
int main()
{
        RGB = 0;
    while (true) {
                if (R == 0) {
                        if (RGB & 1 == 1)
                                RGB = RGB & 6;
                        else
                                RGB = RGB | 1;
                }
                if (G == 0) {
                        if (RGB & 2 == 2)
                                RGB = RGB & 5;
                        else
                                RGB = RGB | 2;
                }
                if (B == 0) {
                        if (RGB & 4 == 4)
                                RGB = RGB & 3;
                        else
                                RGB = RGB | 4;
                }
    }
}
```

4.6 Relays

4.7 Optical Isolators

5 Analog to Digital Conversion

To test the basic idea of a Analog to Digital Conversion (ADC), we can use the concept of a voltmeter. Another way is with a thermistor circuit. Some temperature sensitive devices have a more calibrated response and we have formulae that can translate the voltage to known temperature scales easily.

5.1 Simple Analog In - Voltmeter

This example sets up two versions of the same general circuit. The first version is the simplest one can make. Often, this version does not scale to real world problems. Real world problems can include large power issues that would damage the microcontroller in the process. The second version compensates for circuits with larger currents by placing a optical isolator in between the test circuit and the microcontroller.

I learned to look up the pin name and its location on the board. There pin names that don't necessarily correspond to the label next pin itself. The A0 pin on the CN9 pin set is actually PA_3. It is easy to make the mistake in calling this pin PA_0. This distinction can lead to massive mistakes and cause to operate the board in a wrong manner.

To demonstrate simple ADC in, I use a simple voltage divider circuit with three resistors. Voltage is proportial to resistance. Therefore, we can measure voltage across a resistor and expect its value. That proportion should change if we can the resistance of one of the other resistors.

Therefore, I establish this measurement across a constant 330 ω resistor. Then, I do similar across a 330 ω potentiometer. We see the voltage go from V_{in}. This helps establish a baseline for comparing the ADC readings.

I built the circuit with parts acquired from [18]. The fun kit [19] provides many of the resistors I need. The UNO Project starter [20] provides some addition parts that I can use later. Also the UNO Project starter provides a Arduino UNO that I can use for comparison. Lastly, I used some competitor products to provide the potentiometers and inductors that I need for future examples.

Also, it is important to note a fix on stdio. Mbed.os by default turns off floating point parameters in stdio/UART[21]. There is a parameter in platform/mbed_app.json. By default, a new mbed project has the value for 'minimum-printf-enable-floating-point' set to false. In order to print out floating point, this value needs to be set to true.

With this, we can see in the example that the voltage read from the potenimeter is pretty close to what a common multimeter reads. There are adjustment factors that we apply to give a rough calibration. For production, it would help to obtain a more calibrated set of values.

We see a similar case with thermister based thermometers. I set up a simple 330ω resistor in series with a thermister out of the fun kit. Here we can measure the voltage across the thermister. We then can measure ratios and offsets for temperature.

$$T = \frac{(mV - 225.0)}{10.0} \tag{1}$$

[18] http://www.rexqualis.com/products/

[19] http://www.rexqualis.com/product/electronics-component-fun-kit-w-power-supply-module-male-to-female-jumper-wire-830-tie-points-breadboard-precision-potentiometer-resistor-for-arduino-raspberry-pi-stm32/

[20] http://www.rexqualis.com/product/uno-project-super-starter-kit-for-arduino-w-uno-r3-development-board-detailed-tutorial/

[21] https://github.com/ARMmbed/mbed-os/blob/master/platform/source/minimal-printf/README.md

5.2 Change Flashing Rate with Potentiometer

5.3 Thermostat

5.4 Light Meter

5.5 Thermo-resistor -Taking the Temperature

This is an example of an Analog in problem

5.6 Sound Level Meter

6 Analog Out

6.1 Fixed Voltage

6.2 Sawtooth

6.3 Sine Wave Generator

6.4 Combining the Capabilites of Each Wave Generator Type

6.5 Buzzer Fun

6.6 Pulse Width Modulated Buzzers

7 Optical Isolation

I remember a lesson from my embedded computing class back in the Spring of 1998. Professors Darrel Vines, Michael Parton, and Mike Giesselman told us over and over about the importance of optical isolation. These professors worked in pulsed power, and I wish I had the wisdom back then to realize just how important this subject is.

Pulsed Power delivers a large amount of current in a very short space and time. The amount of current involved can easily melt a microcontroller and destroy it. Yet, these controllers are essential in regulating such power elements to fulfill their purpose.

The fun kit comes with a 4N35 white paper can be found at[22]. The inputs on pins 1 and 2 can be thought of as in and out for a light emitting diode (LED). The light from the LED feeds a photo-transistor that can handle much higher currents.

8 Transistor Circuits

PN2222

[22]https://www.digchip.com/datasheets/parts/datasheet/161/4N35-pdf.php

Transistor

9 Ethernet Web Server Libraries

[23]

10 Digital Signal Processing

[24]

11 Secure Digital Block Devices

12 Home Made RADAR

13 Registering the Serial Number

14 Multiplexed LED Multi-Thread

The original work for this example comes from [**?**]. The value in this exercise exists in the multi-thread operations. We use this lesson to construct classes that inherently use threads to gather information.

[23] https://github.com/khoih-prog/EthernetWebServer_STM32
[24] http://www.emcu.it/STM32F4xx/STM32F4-Library/STM32F4-Library.html