

Creating Alerts with the Arduino Lenardo

Application Note

Lanea Williamson

November 13, 2015

The Arduino microcontroller is a tool used for building prototypes requiring logic. This logic is used to drive the communication between the inputs to the system and the output. There is an abundance amount of support in the Arduino community. The community is able to provide quality resources to accomplish many things. It includes many typical software and hardware interfaces for use in any project. It only requires basic knowledge of the C++ language for usage.

Table of Contents

Objective	3
Introduction	3
Basics of Arduino Programming	4
Variable	6
Interfacing with External Hardware	6
Creating Custom Alerts.....	7
Testing.....	7
Conclusion.....	8
References	9

Objective

The objective of this application note will detail the steps required for building an alarm system using Arduino. There are many things to consider for operation of a microcontroller project which this application note will detail. This note will use an 8-ohm speaker and LED lights for alerts. Basic knowledge of C++ is assumed in the construction of this note. An understanding of software flow and basic C++ keywords is essential for explanation of concepts without providing excessive detail.

Keywords

Arduino, C++, hardware, software

Introduction

The Arduino platform is an open-source prototyping platform based on easy-to-use hardware and software. Arduino Leonardo is a common hardware interface which this note will reference for all hardware-specific topics. Every design concept can be transferred to other Arduino versions, with differences related to number of ports and pin numbers. The Leonardo includes 20 digital input/output pins (12 analog input pins and 7 PWM outputs), and is powered by the ATmega32u4. There are many things to consider when developing an Arduino-based system, including how to handle the interface with the laser range finder and the LED lights and the development of custom visual warnings. On top of these external interactions, consideration needs to be taken when determining where to locate variable declaration. Lastly, a quick overview of testing techniques will be mentioned.

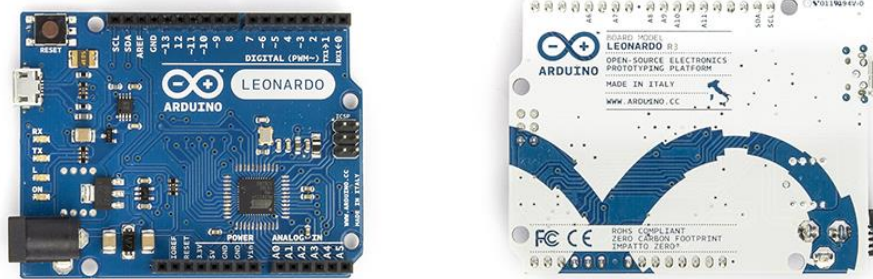


Figure 1: Arduino Leonardo board showing analog I/O pins and digital pins

Basics of Arduino Programming

The basic structure of Arduino programming language runs in at least two parts. All three blocks can be seen in Figure 2. The first block is found at the beginning of the file. First, any *#include* statements will be listed. In this example, we are including a library that will be used for easy interaction to play a melody. Following this, *#define* statements are listed. The ones displayed are for the included library and should be defined for proper duplication of results. Lastly, any global variables are declared.

The second code block is the **void setup()** function. When starting the Arduino, this function is called once. This is where any global variable initialization should occur such as pin modes, or begin serial, as well as any other single-time setup. All parts of this code will be explained in further sections, and the code provided should be referred back to as needed.

After calling the `setup()` function, the `loop()` function does precisely what its name says, and loops consecutively, allowing the program to change, respond, and control the Arduino board. Once the Arduino has finished running the setup function it enters the third code block, the **void loop()** function. Until power is removed from the device, it will continue looping through this function. As with the setup function, further discussion on the function content specifics will be found later in this note.

In addition to these three, a programmer is free to define more functions and refer to them as desired in the loop and setup functions.

```

/*
  Melody
  Plays a melody
  circuit:
  * 8-ohm speaker on digital pin 8
  */
#include "pitches.h"

// notes in the melody:
int melody[] = {
  NOTE_C4, NOTE_C4, NOTE_C4, NOTE_C4, NOTE_C4, 0, NOTE_C4, NOTE_C4
};

// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations[] = {
  4, 4, 4, 4, 4, 4, 4, 4
};

void loop () {
  // iterate over the notes of the melody:
  for (int thisNote = 0; thisNote < 8; thisNote++) {

    // to calculate the note duration, take one second
    // divided by the note type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000 / noteDurations[thisNote];
    tone(8, melody[thisNote], noteDuration);

    // to distinguish the notes, set a minimum time between them.
    // the note's duration + 30% seems to work well:
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);

  }
}

```

Figure 2: Example program used for note

Variable

Basic Arduino programming will not extend into the realm of creating classes. In this case, the functions we created are utility functions, limiting their scope. This is called a local variable which is defined inside the function or as part of a for loop. With this in mind, any variables declared in a loop function are accessible only in that function. The variable *thisNote* declared in the loop function cannot be accessed in other functions. The only way to access this data is to pass the variable directly to the function (as is done in the case of calling *pauseBetweenNotes*). The other limitation in this is that any new entrance into a function deletes the data previously held in scope to that function.

There is another way to persist data – global variables. In our example, *melody* and *noteDurations* are global variables, which prevents it from being overwritten when the loop restarts. A global variable is one that can be seen and used by every function and statement in the program. This variable is declared at the beginning of the program, before the *setup()* function.

Interfacing with External Hardware

The Arduino is powerful, but without external hardware it is very limited in its capabilities. There are many pieces of capable hardware that it can be interfaced with, typically requiring a serial connection. Serial connections do communication across one pin, transferring data in a series of high and low logic inputs. Communications across serial can be tricky, however, since it does not just require checking the current value of a given pin. Thankfully, the *Pitches* library that we included handles the features of this communication, requiring us to make only simple calls to the library in order to receive the data desired.

In the *loop()* function, a large amount of interaction is accomplished using the *Pitches* library. In summary, a command is sent to the range finder to request a tone. Once it receives an acknowledgement, it begins to wait for the data to be received. Once it has received good data, it passes this information back to the *loop* function to be used and processed.

With these few lines of code, full functionality of the melody has been accomplished. It will be discovered repeatedly by any Arduino programmer that many

Libraries can be found that are incredibly useful, especially when interfacing with external hardware. It is a good idea to begin a project with a query for open-source libraries that can accomplish one or more parts of the preferred functionality.

Creating Custom Alerts

In addition to complex hardware like the melody player, it is also easy to work with simple hardware such as LED lights. The last line of the setup function is the only setup needed for preparing to use an LED light. Once plugged into the Arduino as required (between pin 13 and ground in this case), the LED can begin to receive high and low signals, turning the light on and off. Usually a resistor is required to limit the voltage across the LED light to make sure it lasts a long time, however, the Leonardo board includes the proper resistor for pin 13, eliminating this necessity.

The loop function demonstrates the required call for activating the LED light: **digitalWrite(13, HIGH)** to provide voltage to the LED. The keyword *HIGH* would be replaced with *LOW* in order to then remove the voltage from this pin, turning the LED back off found in figure 3.

This functionality can be extended in order to limit the brightness of the LED. By creating a loop in which the voltage is rapidly set to HIGH and then LOW (pulse width modulation), the brightness can be varied by changing the ratio of the duration between each call.

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000); // wait for a second
  digitalWrite(13, LOW); // turn the LED off by making the voltage LOW
  delay(1000); // wait for a second
}
```

Figure 3: Example LED program used for note

Testing

As is the case for any programming accomplished, thorough testing is of utmost importance to ensure functionality of the design. It is wise to spend the most time testing the boundary cases, for example how the system reacts when it changes from an alarm state to a non-alarm state and vice versa. Also test to ensure the melody for alarms is properly selected.

A necessary test for this system would be to determine how much time occurs between notes. Each statement will delay the note further, affecting the time between the tones. This is typically not an issue, but in this case changes the tone speed that is detected by the loop() function. As the delay increases, the time between notes increases as well. There are advanced techniques that can be used to set the time delay between notes to increase the precision, however that is beyond the scope and purpose of this note.

Conclusion

Very powerful hardware and software projects can be developed using the Arduino platform with simple programming. The vast expanse of support through libraries, community forums, and hardware designed for the Arduino make it a top candidate for any project. Arduino programming is a skill that should be mastered by anyone involved in or interested in small scale projects of any sort or those entering into the world of microcontrollers.

References

<http://www.arduino.cc/>