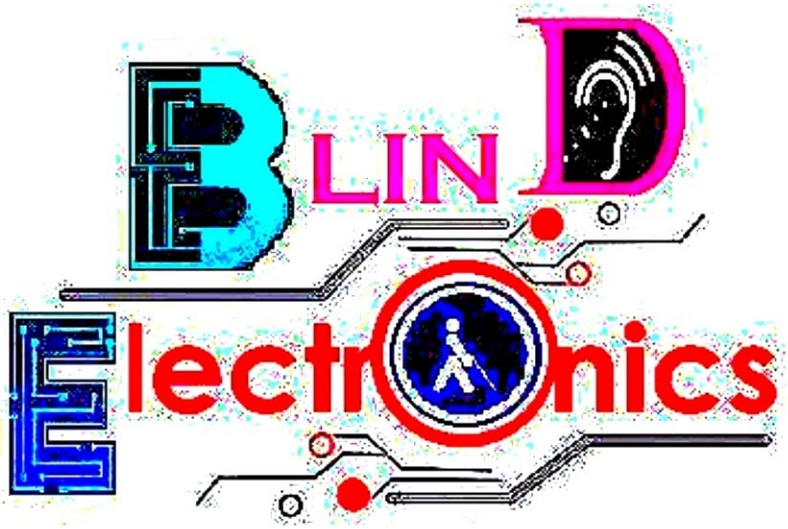

"Everything
should be made
as simple as possible,
but not any simpler."
-Albert Einstein

ASSISTIVE TECHNOLOGY COURSE

Project - Blind Electronics

Guide

Dr. Anil Prabhakar

Dr. Subir Bhaduri

Dr. Amith Nair

Mentor

Dr. Namita Jacob

Dr. Akila Surendran

Presented By

Mr. Sanchit Ghule

Indian Institute of Technology Madras

IITM

publisher

Abstract

The writer wishes it to be clearly understood that the descriptions provided in the following pages, I lays no claim to infallibility. I have written as I have understood, And I have communicated my own understanding of the existing knowledge, which could be improved with different sets of skills and experiences in the journey - however ill-perceived and ill-conceived - to others working along the same path, I claims nothing more. It is just my honest attempt trying to propose a viable path for a blind enthusiast and tinkerers to be able to work in the glorious world of electronics.

> “Does the walker choose the path, or the path the walker?” –
> Garth Nix, Sabriel

0.1 Introduction

Engineering is all about contriving solutions to problems in a creative way. According to my American Heritage Dictionary, the word "engineering" and "ingenious" have a common Latin root, "ingenium," which means talent or skill. In medieval Latin, "ingeniare" was to contrive. The question arises in every conscious engineer's mind is as follows:
Who are the people we serve? How can we properly assess their needs? Given identifiable needs, what is good engineering? How can we expedite the "trial and error" process of optimizing the solutions? Can one engineering staff adequately take on this responsibility?

> "If we do not change direction, we may end up where we are heading" – Siddhārtha Gautama

India with the large population of low vision and blind persons, there is a huge shortage for learning skills and employment opportunities. How could we ensure that differently-abled population gets equal opportunities in all aspects? How do we make sure that they will get appropriate education? is there any educational program in STEM specifically designed for blind and low vision? Particularly what about electronics? Blind Electronics? Is it possible?

The blind and visually impaired community is very much the loser for not sharing individual rehabilitation engineering solutions amongst themselves. They always considered as a consumer or just a user of assistive technology, whether it's electronics, softwares / apps etc. Nobody really cares to make them independent. But we are going to change this scenario by making them equally participating in the resource development process and ultimately making them a self-reliant.

"We cannot blame society's lack of understanding for our ignorance; we must blame our careless isolation which we have allowed to happen. We have isolated ourselves by not sharing our experiences ad by not working toward a collective understanding and identity."

Together, Let us create A Better World!

0.2 Blind Electronics

In order to understand things in gory detail, extensive training and accessible description of some of the electronics component has been created.

Chapter 1

1.1 Blind soldering

How to make iron soldering trusted enough so that VI person will not burn himself or burn down entire house. There should be some sensible way to do Blind Soldering!

In order for "BlindSoldering" to take place, the connection must be hot! hot! hot!, about 550oF (288oC). The solder must be completely liquefied, and the alloying temperature must be reached where the solder "dissolves" or forms an alloy with the metals being bonded. The formation of this alloy on the surface of the metals is called "wetting" of the surfaces by the molten solder. Metals that are electrically active (in the molecular sense) love to combine with oxygen to form oxides. No matter how well we clean the surface of metals, such as copper, it's oxides will form as soon as we stop cleaning. Furthermore, the formation of oxides is tremendously enhanced with the application of heat to the metal. These oxides prevent the Metallurgical Process of Soldering. the molten solder cannot contact the bare metal surface to alloy with it (no wetting occurs). There are agents which remove the oxides as the work is being heated and soldered. These compounds (which are called "fluxes") perform a reducing action which separates the oxides from the surface metal. The flux residue, containing the oxides, floats to the top of the boiling solder and allows the metal-to-metal alloy to form underneath. "Rosin", as used in

soldering, is an inert polymer (complex chain of molecules) which resists forming chemical bonds when it's temperature is lower than the required soldering temperature. When the resin flux is heated, the polymers break up, leaving active molecules to serve as the reducing agent. Upon cooling, the resin and oxide compounds are once again inert, and harmlessly remain on the surface of the joint.

The Basic Soldering Guide - Smith College

www.science.smith.edu/Courses/Readings/Soldering-Guide.pdf

There are as many different soldering systems as there are fabrication processes and materials. Fortunately I came across few cheap hacks of battery powered quick heat irons. Most of the batteries last a few weeks and then they die, or have to keep replacing the tips because they quickly burn out.
The best one is RS PRO Battery Soldering Iron.
RS PRO Battery Soldering Iron, 15W is the iron with fastest heating tip.
very importantly, it's tip heats up very quickly indeed. So, having once placed the tip against the joint we want to solder, by using touch while the tip is cold, we are not going to have to wait so long for the solder to melt so that there is an excessive transfer of heat to the work. this Iron reaches temperature in 7 seconds.
RS Components is not really a consumer site, more selling to trade. Hence their habits of doing stuff like selling a rechargeable soldering iron without the charger.

1.1.1 Solderless Bread Boards

Solderless breadboards have been around for a long time and are designed to be used with components that have leads such as DIP style ICs and leaded resistors and capacitors. In many ways they seem old school in today's world where most components are going Surface Mount Technology (SMT). While SMT components are better in almost every other

way compared to leaded parts, there is no easy way to directly prototype with these newer SMT devices without designing a PCB to mount them. For that reason and for the foreseeable future many of the new and more interesting SMT devices will continue to be adapted for use with solderless breadboards by mounting them on breakout boards of some type so that hobbyists don't need to design and fabricate a PCB every time they want to work with a circuit.

There are numerous opportunities to explorer and make use of this technology for blind hobbyists and blind tinkerers.

www.elenco.com/

1.2 Microcontrollers And Microprocessors

- **Arduino Nano**
- **Raspberry Pi**

description: accessible electronics for blind and visually impaired

Male jumper ends have a pin protruding and can plug into things, while female jumber ends do not and are used to plug things into. female jumper wires which does not have long pins at the end point.

There are 3 types of wires, male to male, male to female and female to female

on the buzzer, we can feel two terminals, one is bigger means positive terminal, other small one means negative terminal. Very careful. pin is just little larger than other pin. its used for PCB thats why it is much smaller

Chapter 2

2.1 Arduino Nano Board Description

After gently passing finger over nano board, we can feel switch and USB port on onside. Just passing finger over the top side of our nano bord, there is a switch which is push down, its for reset purpose, if the board gets hang or freezes while programming.

If we consider USB connecter as the reference and its point towards the computer, Or, Just holding arduino nano in a way that USB connecter is on North Side. (Remember! in maps, North is always on top side of the chart.) Next thing is we can feel two columns of stoke pin-outs on both side of the board.
right side of usb connecter 15 pins.
left side also 15 pins

2.1.1 Arduino Nano Board Description in Table Lawet

Pin Name	Count	Pin Name
Left Side	USB Socket	Right Side
D13	1	D12
3v Out	2	D11
REF	3	D10
A0	4	D9
A1	5	D8
A2	6	D7
A3	7	D6
A4	8	D5
A5	9	D4
A6	10	D3
A7	11	D2
5v Out	12	GND
RST	13	RST
GND	14	RX0
VIN	15	TX0

right side of USB connecter pin starts from D12, D11, D10, D9, D8, D7, D6, D5, D4, D3, D2, GND, RST, RX0, TX0

"D" referred as a digital pin.

GND = Ground

xRST = Reset

RX0 = Serial Receiver

TX0 = Serial Transmitter

left side of usb connecter pins start from D13, 3v, REF, A0, A1, A2, A3, A4, A5, A6, A7, 5v, RST, GND, VIN

"A" referred as an analog pins.

2.1.2 Arduino IDE or Commandline

Using Arduino-mk

The default development IDE for Arduino is no doubt very powerful, but I prefer to use the command-line. The reasons for this are several:

- Most compelling is the fact that I am blind and the Arduino IDE is not enough accessible.
- I am a command-line junkie.
- Using the AVR version of the gcc tool-chain in the usual gcc way makes me feel closer to the hardware.
- The graphical tools use the same tool-chain anyway

now onwards 'command-line' will abbreviated as 'cli'.

In this tutorial I am going to describe installing the core Arduino tools on a Debian Linux machine, and how to use the 'Arduino-mk' tool to hack stuff at the cli. Adjust the commands below for wer package-manager and package names if we aren't using Debian. Installing the Tools First we are going to need to install the core Arduino development tools.
we will need the 'build-essential' package that includes all the Gnu binutils including make etc.
we may already have this if we are a coder:
$ sudo apt-get install build-essential
Now need to install the Arduino stuff:
$ sudo apt-get install arduino arduino-core arduino-mk
Those three packages will pull in everything we need:

- GUI IDE and the needed Java run-time.
- Core Arduino libraries etc., and the AVR tool-chain.
- Arduino-mk, our first cli tool.

What is Arduino-mk?

Arduino-mk is a generic Makefile which conforms to the Gnu make syntax and which we can include into any Makefile we construct to build our code from the cli.

Hello World, but with no Display

It's usual for any first project written in a language we are learning to be a 'Hello World!' project. Typically a program that does nothing but print 'Hello World!' to the screen and exit, or wait for an exiting button-click.
But we don't have a screen on the Arduino. So we are going to say a virtual 'Hello World!' over and over again by making an LED blink on and off in perpetuum, or at least 'til we pull the plug.
It is standard on an Arduino for there to be an on-board LED attached to pin 13, and by pulsing this pin we can make the LED blink.
Because I can't see and because this is an audio medium, I'm also going to connect a small 3 volt piezo-electric buzzer to pin 13 so we can hear that it is working.
In Arduino world, a project is usually called a 'sketch'. I can't help finding this a little patronising, but we will stick with it for the moment.
An Arduino sketch, or source file has the extension .ino. The Arduino tools will convert the code into C++ before it is compiled.
The .*ino* code looks a lot like **Vanilla C**.

To set up our project we follow these steps:

1. Creating a directory

   the build process will yield two files which have the same name as the directory and the extensions .elf and .hex.
   So, first we make our directory and cd to it:
   $ mkdir blink1
   $ cd blink1
   And here is the source listing for blink.ino:

```
#define LED_PIN 13

void setup()
{
pinMode(LED_PIN, OUTPUT);
}

void loop()
{
digitalWrite(LED_PIN, HIGH);
delay(100);
digitalWrite(LED_PIN, LOW);
delay(900);
}
```

we may notice there are two functions; setup() and loop(). These two functions always exist in an Arduino sketch.
The setup() function is called once at the beginning of execution, and then the loop() function does just what it says on the tin, it loops until power-down.
But I can't see where either of these functions are called from...
That's because the build process will add a minimal main() function for us which calls first the setup() function and then the loop() function.
What this code does is as follows:

(a) Defines a constant called LED_{PIN} with a value of 13

(b) In the setup() function initialises pin 13 as an output

(c) In the loop() function:

(d) Set pin 13 high (LED on)

(e) Wait 100 milliseconds

(f) Set pin 13 low (LED off)

(g) Wait 900 milliseconds

(h) Go to 4

2. The Makefile

   Now we will write a Makefile with which we can compile and ultimately upload this program. Here is the source of the Makefile:

```
ARDUINO_DIR  = /usr/share/arduino

#ARDUINO_LIBS = Ethernet Ethernet/utility SPI
BOARD_TAG    = nano
ARDUINO_PORT = /dev/ttyACM0

include /usr/share/arduino/Arduino.mk
```

   Each of these directives should be self-explanatory. I have commented out the $\text{ARDUINO}_{\text{LIBS}}$ line because our simple blink project does not use any libs but I have left it in to illustrate where to name extra libs.
   The $\text{BOARD}_{\text{TAG}}$ directive is set to 'nano' because that is the type of hardware we are using.
   The crucial bit is the include line at the bottom. This includes the generic Makefile which was installed when we selected the arduino-mk package for installation.

3. Build

   Now we can build this project with command:
   $make
   we will see a warning fly past about depends.mk not existing. This file will be created on the first compile.
   If this all works there will now be a sub-directory called 'build-cli', in which there are a whole bunch of files.

There are loads of *.o files, including a lot which bear no resemblance to any source we created. These are standard library files which have been compiled and which are linked into the result.
The files which are crucial to us now are:

- blink1.elf
- blink1.hex

Note that even if wer source file was called blink.ino, these two files will be named the same as the directory in which we are working. The .hex file is an ASCII file which contains a dump of the op codes written into the .elf file (I think). If we now use the Linux 'file' command to inspect blink1.elf, we get this result:
$blink1.elf: ELF 32-bit LSB executable, Atmel AVR 8-bit, version 1 (SYSV), statically linked, not stripped
As we can see, this is an Atmel 8-bit executable, not a Linux executable.
The blink.hex file is ASCII but it is the values in this file which get written into the programmable chip on the Arduino board.

4. Upload

   To upload the compiled code to the Arduino:
   $ sudo make upload
   We need to use sudo for this to get permissions to the serial device. Obviously the Arduino needs to be connected to wer Linux machine via USB. The USB port will also provide power. When we connect the Arduino to wer machine, check the allocation of the port device like this:
   $ dmesg tail
   we should see which device has been allocated. we could also:
   $ ls /dev/ttyACM*
   I think the port will be different on a Mac. And of course if we have more than one Arduino plugged in, the ports will be numbered; /dev/ttyACM0, /dev/ttyACM1 etc.

Once the upload is complete, press the reset switch of the nano board and the LED will, if all was well, begin to flash.

A Final Note

What we have just done is ‘cross-compiled’ an executable. That is we have compiled some code on one platform, for running on another platform, or ‘flavour’ of processor.

Using Ino

In the first part of this tutorial we installed the core Arduino development tools and the first of the tools we were using for working at the cli.
Now we are going to install another tool; ino.

To recap, in the last part we did these installs:

$ sudo apt-get install build-essential
$ sudo apt-get install arduino arduino-core arduino-mk

Now we need some other bits and pieces, some of which, for example Python2.7, we may already have:

$ sudo apt-get install python2.7
$ sudo apt-get install python-pip
$ sudo pip install ino

What is ino?

ino is a tool written in Python. While it ultimately uses the same AVR tools that are used by the Arduino IDE, and by the Arduino-mk tool we used last time, it offers a different way of achieving the same results.
Where ino wins over other cli tools is that it hides the messy bits of project authoring and building from the user.
Personally, I like seeing all the messy stuff.
ino also includes some example and skeleton projects.
The source code for 'blink', which we built in the first part of this tutorial using Arduino-mk was actually pulled from one of the ino example projects.
So, now let's use ino to create and build the same project.
As I did last time, I am going to connect a small piezo-electric buzzer to pin 13 so that we can hear that it is working.
Follow these steps:

Creating a directory for it.

$ mkdir blink2 $ cd blink2

Now we use ino to initialise the blink project from one of the ino examples:

$ ino init -t blink

This will create two sub-directories in our project directory:
lib/
src/
In the src directory there is one source file, or ‘sketch’, called:
sketch.ino
If we looked at the source code we would see it is exactly the same as the blink.ino source file we created for the blink1 project in the first part of this tutorial.
In the lib directory there is one file, called .holder, which appears to be empty.
We are not using any libraries other than the defaults in this so there will be nothing in lib.
Now we build the project:
$ ino build
This will create a directory called .build (see what I mean about ino hiding the good stuff?).
In .build there are two things now:
environment.pickle
nano/
The .pickle file is a Python mechanism for taking a snap-shot of the build environment at build time.
The directory nano, is where we will find the meat.
Note that the nano directory is only called nano because it is an Arduino nano for which we are developing, the default board type is uno.
If we chose to use a different Arduino version, this directory would be named the same way as the board type.
Looking in the nano directory we see a bunch of stuff:
arduino/
firmware.elf*
firmware.hex
Makefile
Makefile.deps
Makefile.sketch

src/
If we looked in arduino/, we could see files which should be familiar to us if we followed the first part of this tutorial, when we built blink using Arduino-mk. There are a whole load of object files, some .d dependancy files and other things.
Now in this directory, we also see:
firmware.elf
firmware.hex
These two files correspond to the blink1.elf and blink1.hex files which were created by Arduino-mk last time.
It is these files which are used in the upload, when we issue this command, in our original project directory:
$ sudo ino upload
Hang on a second... some of the messages that flew past looked suspiciously similar, or even identical to what happened when we used our Arduino-mk Makefile to upload blink1 in the first part of this tutorial.
Well yes, all that both Arduino-mk and ino are doing in both parts of this tutorial is to call the same Arduino AVR tools.
Again, press that reset button after the upload and the LED on pin 13 will start to blink.

Arduino Uno Board Description and Pinouts

www.raspberryvi.org/posts/arduino-uno-board-description

Pull-up and pull-down Resistors

www.raspberryvi.org/posts/pull-up-and-pull-down-resistors

2.1.3 Raspberry Pi

Following is the perfect Raspberry Pi board description wrote by Mike Ray www.raspberryvi.org/ is the best website for most of the opensource hardware, DIY, and how to do it? Good stuff.

- Raspberry Pi Board Description (All Models)

www.raspberryvi.org/pages/guides/board-description

2.1.4 Other Hardware

of course, we're not limited only to a pi, there's lots and lots of single board computers (SBC) devices that can do any number of tasks.

- Propeller Project Board
- Beaglebone Black
- Intel NUC Mini PC
- Intel Computestick
- Atmel-11269-32-bit-Cortex-A5-Microcontroller-SAMA 5D3 breadboard

Chapter 3

3.1 Reconciling Re-Search

This would be a continuem Re-Search examining the use of the blind electronics as a adjunct to classroom techniques.
Many more things to come to built either on top of Linux or Android, and this will propel us into the future where more modern hardware can be used within these devices, so Linux libraries which drive some of this now, which opens up a potential of re-use or easier re-configuration to fit a wide range of hardware solutions.

De dilemma arises out of the paradoxical question:
Should our decision precede our action/execution or our action precede our decision?
That might probably answer if any decision taken is wise or otherwise!
Maybe this is just a Memorising and text torturing. Dssssssss?

So The Linux revolution is already here, and we live among it.

> "We're not lost. We're just headed somewhere different." – Emily X.R.

3.2 Future Work

The future goal of blind electronics project is to create accessible electronics hardware and software as well as creating accessible online electronics learning material for all people with depraved vision, blind and

deaf blind people, at the same time, to open the online market for the exchange of various accessible models and applications between blind people and people with low vision, their parents and teachers, as well as programmers who would develop accessible programs/solutions for them. All major arduino family microcontrollers can be made accessible by using some special shield by extending GPIO headers. This special shield would make life easy for VI person to put bits and pieces together.

www.aph.org/product/snapino-access-kit/

Raspberry Pi OS can be made accessible for blind and low vision, in fact special Accessible Operating System can be made out of Linux Kernel. Special shield for GPIO headers along with external components such as servomotor, LEDs and other sensors can be put together and accessible kit can be made.

the GoBraille is built on a Raspberry Pi.

Linux does have a huge place to carve out future of assistive technology. here is some documentation on Accessibility Debian WIKI and Accessibility Arch Linux Wiki.

3.3 Other Components

Most of the commonly used hobbyist hardware will be examined and made accessible for VI persons use.

Name	link
Axis Accelerometer & Gyroscope	https://www.seeedstudio.com/Grove-6-Ax
Adjustable PIR Motion Sensor	https://www.seeedstudio.com/Grove-Adjus
Dual Button	https://www.seeedstudio.com/Grove-Dual-
Light Sensor	https://www.seeedstudio.com/Grove-Light
Passive Buzzer	https://www.seeedstudio.com/Grove-Passi
Rotary Angle Sensor (Rotary Potentiometer)	https://www.seeedstudio.com/Grove-Rotar
Servo	https://www.seeedstudio.com/Grove-Servo
Speaker	https://www.seeedstudio.com/Grove-Speak
Ultrasonic Distance Sensor	https://www.seeedstudio.com/Grove-Ultra
Universal 4 Pin Buckled 30cm Cable	https://www.seeedstudio.com/Grove-Unive
Shield for Pi Pico	https://www.seeedstudio.com/Grove-Shiel
Raspberry Pi Pico Pre-Soldered	https://www.seeedstudio.com/Raspberry-P

3.4 Shortcomings

Making electronics hardware accessible for blind users is painful task for any manufacturing industry. In addition to that the clever clause may not allow us to invest so much of time and money, so How do we ensure that we're rolling out accessible, commercially suitable, economical and sustainable hardware product.
Manufacturing Industry : PL] Don't be in such a hog! blind people have to use it!

3.5 References

Smith-Kettlewell Technical File

www.ski.org/smith-kettlewell-technical-file

Blind Arduino Blog

http://blarbl.blogspot.com/

The Making of a Blind Maker

https://nfb.org//sites/www.nfb.org/files/images/nfb/publications/fr/fr35/2/fr350205.htm

DIY Ability, Makers With & Without Disabilities

www.diyability.org/

NASA - Blind Students Rocket On!

www.nasa.gov/audience/foreducators/9-12/features/F$_{\text{RocketOnNFB}}$

Project Math Access

www.tsbvi.edu/math/

Challenges and solutions when using technologies in the classroom

https://files.eric.ed.gov/fulltext/ED577147.pdf

“So at the end of this day, we give thanks For being betrothed to the unknown.” – John O’Donohue