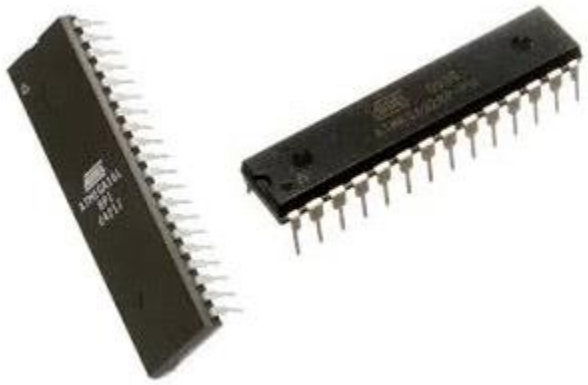
WHAT IS AVR MICROCONTROLLER?

An AVR microcontroller is a type of device manufactured by Atmel, which has particular benefits over other common chips, but first what is a microcontroller?

The easiest way of thinking about it is to compare a microcontroller with your PC, which has a motherboard in it. On that motherboard is a microprocessor (Intel, AMD chips) that provides the intelligence, RAM and EEPROM memories and interfaces to rest of system, like serial ports (mostly USB ports now), disk drives and display interfaces.

A microcontroller has all or most of these features built-in to a single chip, so it doesn't need a motherboard and many components, LEDs for example, can be connected directly to the AVR. If you tried this with a microprocessor, bang!

AVR microntrollers come in different packages,some designed for through-hole mounting and some surface mount. AVRs are available with 8-pins to 100-pins, although anything 64-pin or over is surface mount only. Most people start with a DIL (Dual In Line) 28-pin chip like the ATmega328 or the 40-pin ATmega16 or ATmega32.



PC microprocessors are always at least 32-bit and commonly now 64-bit. This means that they can process data in 32-bit or 64-bit chunks as they are connected to data buses this wide. The AVR is much simpler and deals with data in 8-bit chunks as its data bus is 8-bit wide, although there is now an AVR32 with 32-bit bus and an ATxmega family with a 16-bit data bus.

A PC has an operating system (Windows or Linux) and this runs programs, such as Word or Internet Explorer or Chrome that do specific things. An 8-bit microcontroller like the AVR doesn't usually have an operating system, although it could run a simple one if required, and instead it just runs a single program.

Just as your PC would be useless if you didn't install any programs, an AVR must have a program installed to be any use. This program is stored in memory built-in to the AVR, not on an external disk drive like a PC. Loading this program into the AVR is done with an AVR programmer, usually when the AVR is in a circuit or system, hence AVR ISP or AVR In System Programmer.

AVR ISP for AVR Microcontroller Programming

So what is a program? A program is a series of instructions, each very simple, that fetch and manipulate data. In most applications where you would use an AVR, such as a washing machine controller for example, this means reading inputs, checking their state and switching on outputs accordingly. Sometimes you may need to modify or manipulate the data, or transmit it to another device, such as an LCD or serial port.

A series of simple binary instructions are used to do these basic tasks and each one has an equivalent assembly language instruction that humans can understand. The most basic way of writing a program for an AVR is to use assembly language (although you could write binary numbers if you want to be pedantic).

Using assembly language allows you to understand far more about the operation of the AVR and how it is put together. It is also produces very small and fast code. The disadvantage is that you as the programmer have to do everything, including memory management and program structure, which can get very tedious.

To avoid this, high level languages are increasingly being used to write programs for the AVR, C in particular but also Basic and Java derivatives. High level means that each line of C (or Basic or Java) code can translate into many lines of assembly language.

The compiler also deals with the program structure and memory management so it is much easier. Commonly used routines, such as delays or maths, can also be stored in libraries and reused very easily. The C compiler also deals with larger numbers that take up more than a byte (8-bits).

In my opinion, writing AVR programs in C is like driving a car. Yes you can do it very easily but if something goes wrong you haven't got a clue how to fix it and you can't deal with tricky situations like icy roads. Starting with assembly language and writing some simple programs lets you understand what is going on "under the hood" so you know how it works and can get the most out of it. Then swap to C by all means but at least you know how the AVR microcontroller fits together and its limitations.

Advantages of using AVR

- You'll get deep understanding how microprocessors work and it's an important gateway to the world of computer engineering
- You'll learn how to code with C
- The major advantages AVR has over Arduino that you can write a program for any other AVR microcontroller. You just need the datasheet of the particular microcontroller. For example, I can write the program for Atmega 16, Atmega 32, Atmega 8, Atmega 328. I may or may not need the development board to program the particular microcontroller. With the little bit of experience, I can develop my own board. With Arduino, I can only write the program for the microcontroller it's using as an MCU.
- AVR's resources is easy to get for purpose of learning !

Time spent to get things done

- 1 – 3 months

Courses could be taken

- https://www.youtube.com/playlist?list=PLD7F7ED1F3505D8D5
- https://www.youtube.com/channel/UCbZ7PLd5LAnje1hpyoiRW0A/playlists
- mazidi's avr microcontroller and embedded systems
- Youtupe it using filter "playlist"

What is Arduino?

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of accessible knowledge that can be of great help to novices and experts alike.

Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and embedded environments. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The software, too, is open-source, and it is growing through the contributions of users worldwide.

Why Arduino?

Thanks to its simple and accessible user experience, Arduino has been used in thousands of different projects and applications. The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux. Teachers and students use it to build low cost scientific instruments, to prove chemistry and physics principles, or to get started with programming and robotics. Designers and architects build interactive prototypes, musicians and artists use it for installations and to experiment with new musical instruments. Makers, of course, use it to build many of the projects exhibited at the Maker Faire, for example. Arduino is a key tool to learn new things. Anyone - children, hobbyists, artists, programmers - can start tinkering just following the step by step instructions of a kit, or sharing ideas online with other members of the Arduino community.

There are many other microcontrollers and microcontroller platforms available for physical computing. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's

Handyboard, and many others offer similar functionality. All of these tools take the messy details of microcontroller programming and wrap it up in an easy-to-use package. Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:

- Inexpensive - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than $50
- Cross-platform - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
- Simple, clear programming environment - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.
- Open source and extensible software - The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.
- Open source and extensible hardware - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

Pros and Cons of Using Arduino

Pros :

- Ready to use
- Examples of code
- Effortless functions
- Large community

Cons :

- Structure : Yes, the structure of Arduino is its disadvantage as well. During building a project you have to make its size as small as possible. But with the big structures ofArduino we have to stick with big sized PCB's. If you are working on a small micro-controller like ATmega8 you can easily make your PCB as small as possible.
- Cost
- Easy to use "you will not a lot of new things"

Time spent :

Maximum one week

Courses :

- Simple Arduino PDF book
- Jeremy blum channel on youtupe
- Micropedia channel on youtupe

Tiva-c

The Tiva-C (a.k.a. TM4C) LaunchPads[1] are inexpensive self-contained, single-board microcontrollers, about the size of a credit card, featuring an ARM Cortex-M4F 32-bit CPU operating at 80 to 120 MHz, manufactured by Texas Instruments.[2] The TM4C Series TM4C123G LaunchPad[3]is an upgrade from TI's Stellaris LaunchPad adding support options for motion control PWMs and USB Host functionality. The more recently released TM4C1294 Connected LaunchPad[4] is the first cloud-connected offering in TI's LaunchPad ecosystem and provides a solid foundation for beginning and evaluating embedded IoT designs.

There are many I/O pins (40 to 80 depending upon version) that have multi-personality. This means that they can be easily configured as

digital inputs or outputs, analog inputs and outputs or other functions, allowing a great variety of applications, are just the multiple serial ports have the ability to interface with more items such as test cards or other communication modules, etc. Among those pins there are included the GND and POWER (3.3 V) pins.

The clock is 80 or 120 MHz (vers based), which makes them 5 to over 7 times faster than the Arduino Uno's 16 MHz ATmega328P microcontroller. As with any Cortex M4, the CPU has some DSP (digital signal processor) instructions, with some limitations. One can do signal processing, for example, sampling a human voice with a good quality, able to be processed in MATLAB.[*citation needed*] The CPU contains the optional floating-point unit with single-precision floating point operations supported.

They have an additional USB port which can act as USB host, allowing the connection of multiple devices and the "Connected" one has an integrated 10/100 Ethernet MAC+PHY for Internet connectivity. They also have a temperature sensor and on-board LED(s) and RGB LED(s), which allows you to generate various colors by combining the three basic colors (red, blue and green) of the additive color synthesis.

The Tiva/TM4C LaunchPads come preloaded with software to demonstrate many of the capabilities of the ARM microcontroller and with a quickstart application to get up and running within minutes.

To summarize why we don’t need to work with it that it takes much time to learn and it’s required to have a background on how microprocessors work and It costs 400 L.E

To sum up : AVR is the optimum solution in my opinion but the difficulty and the challenge will be in doing it on pcb and 1 mistake could throw out most of our efforts

So the plan B in this case that we will be divided as some do the work using Arduino and the others will do it using AVR