Getting started with the Pico

The new raspberry pi pico is a super powerful micro-controller , this book intends to explore just some of it's capabilities

- Introduction to the toolchain
  - Installing everything
  - Introduction to Make and Cmake

Introduction to the toolchain

Setting up the toolchain and the pico-sdk and getting introduced to it

Installing everything

Install the arm toolchain

```
sudo apt update
sudo apt install cmake gcc-arm-none-eabi libnewlib-arm-none-eabi build-essential
```

Get yourself the pico-sdk

SO the good folks at the raspi-foundation have kind of compiled an sdk for the users to easily compile and build programms for the pico So to get the SDK you have to clone the repo

```
 git clone -b master https://github.com/raspberrypi/pico-sdk.git
 cd pico-sdk
 git submodule update --init
 cd ..
 git clone -b master https://github.com/raspberrypi/pico-examples.git
```

Get the examples

Now let us clone the examples that we are going to use for the rest of this book

```
https://github.com/raspberrypi/pico-examples
```

That's it for the setup

Introduction to Make and Cmake

Intro to Make and Cmake

Let's start simply by compiling the below cpp programm

```
#include <iostream>


int main()
{
    std::cout << "Hello World";
}
```

```
g++ ./hello.cpp
```

This would give us an a.out file which we can run as a binary file. Now this was a simple step, just run one command to generate the binary file . What if I ask you that I don't need just this one file but I need multiple others as well such as .hex file or maybe a .uf2 file along with it . You would probably then run multiple commands to generate those other kinds of files.

GNU make and automation

This is where make comes in, make basically lets you automate a lot of things just by including a makefile in your project and adding kind of verbs in that .

Allow me to demonstrate

Now in the previous directory add a file called Makefile with these contents

```
compile:
	g++ ./hello.cpp
```

now run install make by using your package manager for example by apt

```
apt install make
```

now try running make compile in the same directory where the makefile is placed. At the end you are going see the same a.out file which instead was generated by g++ earlier.

Now edit your makefile to generate an object file as well

```
compile:
	g++ ./hello.cpp
all:
	g++ -c ./hello.cpp
```

Now if you run `make all` you will see that a hello.o ie an object file has also been generated. If you want to get knee deep into Makefiles head here

Now that we have a grasp on makefiles let's move to Cmake

Cmake

Now lets add another file to our directory called CMakeLists.txt

```
 cmake_minimum_required(VERSION 2.8.9)
project (hello)
add_executable(hello hello.cpp)
```

Now remove the ./a.out file and create a folder called build head to the build folder and run

```
cmake ..
```

You are going to see some output and then a bunch of files generated in the folder. One of these is a Makefile ! Now if you open this Makefille , you will find a beautifully generated Makefile with explicit instructions on how to compile it. Now run

```
make all
```

This will generate a binary called hello in the same directory and if you run this binary

```
./hello
```

You are going to see the hello world printed which was supposed to be there .

Explaination of various directives in the CMakeLists.txt file

- The first line sets the minimum required cmake version for this project

```
cmake_minimum_required(VERSION 2.8.9)
```

- The second line specifies the name of the project

```
project (hello)
```

- And the third and the most important third line adds an executable named hello and generates it from the hello.cpp file.

```
add_executable(hello helloworld.cpp)
```

Note : Here is more information about cmake
