InfiniTime

Release 1.7

InfiniTime

Mar 03, 2022

CONTENTS

CHAPTER

ONE

WHAT IS INFINITIME

1.1 Generalities about PineTime

PineTime is an open source smartwatch produced by Pine64.

From Pine64's website:

"The PineTime is a free and open source smartwatch capable of running custom-built open operating systems. Some of the notable features include a heart rate monitor, a week-long battery, and a capacitive touch IPS display that is legible in direct sunlight. It is a fully community driven side-project which anyone can contribute to, allowing you to keep control of your device."

More details can be found on the PineTime page of the Pine64 website.

1.2 InfiniTime goals

The goal of this project is to design an open-source firmware for the Pinetime smartwatch :

- Code written in **modern C++**
- Build system based on **CMake**
- Based on FreeRTOS 10.0.0 real-time OS
- Using LittleVGL/LVGL 7 as UI library
- and NimBLE 1.3.0 as BLE stack

1.3 InfiniTime features - high level

InfiniTime implements the following features:

- Rich user interface via display, touchscreen and pushbutton
- Time synchronization via Bluetooth Low Energy (BLE)
- Time setup via the watch itself
- Notifications via BLE
- Heart rate measurements
- Steps counting
- Wake-up on wrist rotation

- Quick actions
  - Disable vibration on notification
  - Brightness settings
  - Flashlight
  - Settings
- 3 watch faces:
  - Digital
  - Analog
  - PineTimeStyle
- Multiple 'apps' :
  - Music (control the playback of music on your phone)
  - Heart rate (measure your heart rate)
  - Navigation (displays navigation instructions coming from the companion app)
  - Notification (displays the last notification received)
  - Paddle (single player pong-like game)
  - Twos (2048 clone game)
  - Stopwatch
  - Alarm App
  - Steps (displays the number of steps taken)
  - Timer (set a countdown timer that will notify you when it expires)
  - Metronome (vibrates to a given bpm with a customizable beats per bar)
- User settings:
  - Display timeout
  - Wake-up condition
  - Time format (12/24h)
  - Default watch face
  - Daily steps goal
  - Battery status
  - Firmware validation
  - System information
- Supported by multiple companion apps (development is in progress):
  - Gadgetbridge (on Android via F-Droid)
  - Amazfish (on SailfishOS and Linux)
  - Siglo (on Linux)
  - ITD (on Linux)
  - [Experimental] WebBLEWatch Synchronize time directly from your web browser. video

  - [Experimental] InfiniLink (on iOS)
  - OTA (Over-the-air) update via BLE
  - Bootloader based on MCUBoot

1.4 InfiniTime history

1.4.1 Project history

The project was started by JF in October 2019. JF already knew Pine64 as a company that builds ARM-based SBCs, and had followed the Pinebook Pro and PinePhone. In September 2019, JF saw an article about the PineTime, an open-source smartwatch, stating that Pine64 was looking for developers to build a software running on it.
After he contacted Pine64 to explain his motivation, he got sent a PineTime dev kit and starting tinkering with it.

The first version of the bootloader, based on MCUBoot, was written by Lup Yuen.

The first versions of InfiniTime were mainly a one-man show, but quickly a lot of developers joined the project, and the speed of development and therefore new features increased a lot:

- The first release, version 0.1.0 already provided basic functionalities : display the time, BLE connection and time synchronization. There was no touch interface, the UI was slow and basic.
- Version 0.2.2 was the first release that received contributions from other developpers :) project!
- Version 0.3 switched from a custom graphics library to LittleVGL, a light and versatile graphics library for embedded systems.
- Version 0.5 introduced a change of the BLE stack: instead of using the NRF SoftDevice (the BLE stack from Nordic Semiconductor), the decision was taken to use NimBLE, an open source BLE stack.
  Amazfish, a companion app for Sailfish OS and Linux, added support for InfiniTime, becoming the first companion app for the smartwatch.
- Version 0.6 brought the OTA functionality, allowing users to upgrade the firmare over-the-air, using their smartphone and BLE connectivity.
- Version 0.7.1, which brought improvements, optimisations, became the official firmware sent with new PineTime (previous firmware was a “work in progress” one, and closed-source).

1.4.2 Firmwares history and changelog

The releases history can be found here on GitHub.

For a list with a summary of new features for each release, please see this article.

CHAPTER

TWO

USER DOCUMENTATION

This part of the documentation is meant for end users.
For developer documentation, please refer to this page

2.1 Getting started with InfiniTime

2.1.1 Unboxing your PineTime

The box contains:

- the PineTime,
- a USB-A charger/craddle,
- a quick user guide.

2.1.2 Boot/Reboot/Switch off InfiniTime

The PineTime has a single button, located on the left hand side.

- To start/boot your PineTime, simply hold the button for a few seconds until the Pine64 logo appears,
  - if nothing happens, you may have to charge the watch beforehand.
- To reboot/restart the watch, press and hold the button for approximately 8 seconds
  - Release the button at that stage, otherwise you will trigger another action (see Recovery firmware and Firmware validation for more details).
- It is **not** possible to switch it off.

2.1.3 Setting up date and time

By default, InfiniTime starts on the digital watchface. It'll probably display the epoch time (1 Jan 1970, 00:00).

You can set the time (and date) manually, or have a companion app do it for you.

InfiniTime doesn't handle daylight savings automatically, so make sure to set the correct time or sync with a companion app.

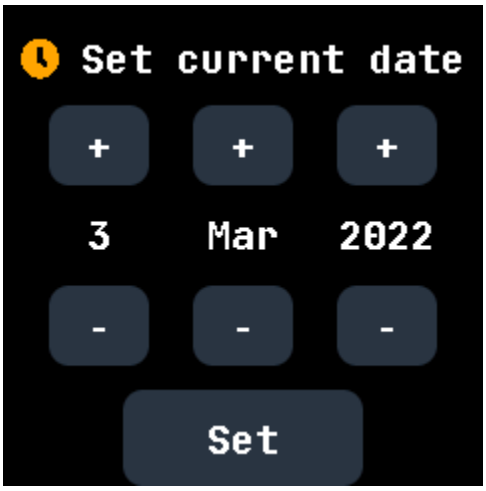
Using companion apps

Date and time are set by the companion app once the PineTime is connected over BLE.

Manually

Starting with InfiniTime 1.7, it is possible to configure the date and time directly from the watch.

This can be done from the Settings menu, then “Set date” and “Set time” respectively:



Using any Chromium-based web-browser

You can use WebBLE from a Chromium-based browser (Chrome, Chromium, Edge, Chrome Android) to setup the date and time.

(insert here pics of WebBLE GH)

Using NRFConnect

You must enable the **CTS** *GATT* server into NRFConnect so that InfiniTime can synchronize the time with your smartphone.

Launch NRFConnect, tap the sandwich button on the top left and select *Configure GATT server*:

Tap *Add service* and select the server configuration *Current Time service*. Tap OK and connect to your PineTime, it should automcatically sync the time once the connection is established!

(insert pics from https://github.com/InfiniTimeOrg/InfiniTime/blob/develop/doc/gettingStarted/ota-gadgetbridge-nrfconnect.md#using-nrfconnect-1)

2.1.4 Companion apps

PineTime can be used as a standalone watch, displaying date and time (which can be configured from the watch itself since InfiniTime 1.7.0), as well as heart-rate, number of steps, used as a torchlight, or just to play the included games (2048, Pong, and Draw).

To get more features, Companion apps, which are applications running on a smartphone or a computer, and are paired to the PineTime, are required.

There are multiple Companion apps available:

- Smartphones:
  - Android: GadgetBridge
  - SailfishOS: Amazfish
  - iOS: InfiniLink
  - PinePhone (Linux phone): Siglo
- Linux Computer:
  - Amazfish and Siglo will also work, but may require some manual installation.
  - ITD

2.1.5 The InfiniTime UI

The UI is composed of 4 different areas:

- the main watchface
- the notification screen (swipe down)
- apps drawer (swipe up)
- quick settings (swipe right)

Watchfaces

The default watchface is the "digital one", which displays date and time, as well as the number of steps, heart-rate, bluetooth icon (when connected), battery status, and possibly missed notifications.

There are 2 other watchfaces:

- Analog
- PineTimeStyle

Notification screen

When swiping down, the last notification is displayed.
Up to 5 notifications can be displayed: simply swipe up again the display the next notification.
To come back to the watchface, press the left button.

It is currently not possible to discard notifications, unless you restart InfiniTime (long press on the button for ~8 seconds).

Apps drawer

When swiping up, the apps drawer allows launching applications.

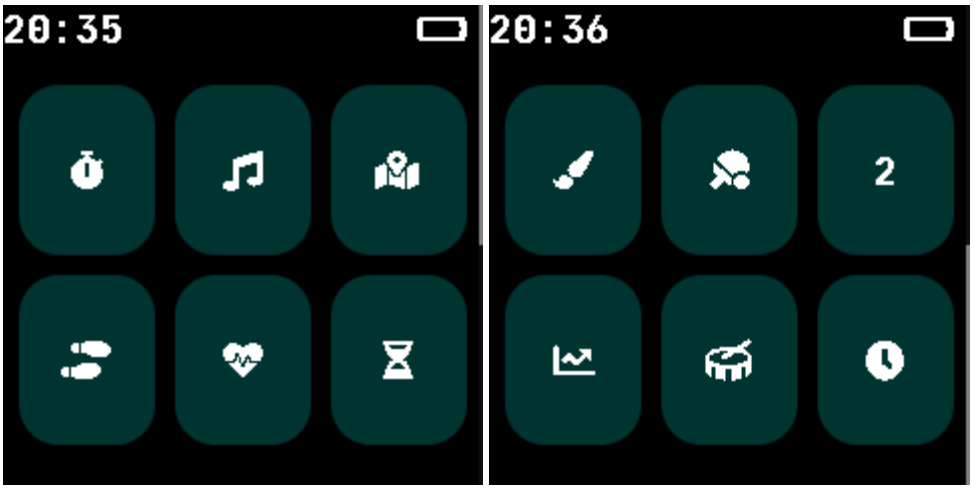
There are 2 pages with each 6 applications (2 rows of 3 apps):

Page 1:

- Stopwatch
- Music control
- Navigation (only works with PureMaps/Sailfish OS)
- Steps counter
- Heart-rate
- Countdown

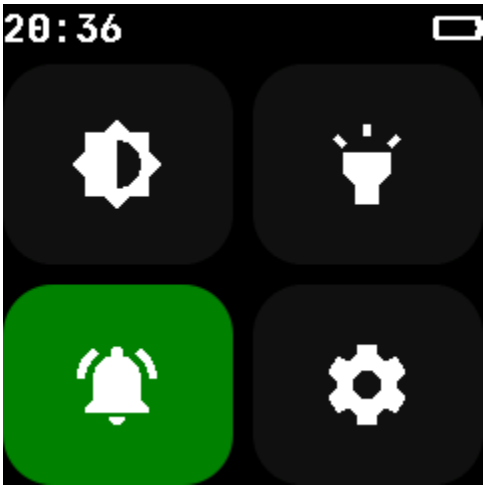
Page 2:

- Draw
- Pong game
- 2048 game
- Accelerometer
- Metronome
- Alarm



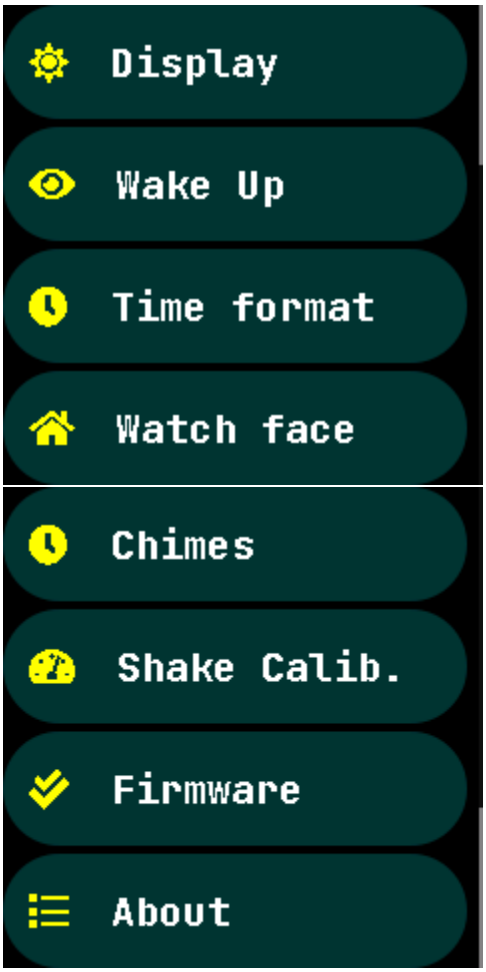
Quick settings

When swiping right, you get access to 4 icons :



- brightness level: pressing it will cycle between 3 levels (low/medium/high)
- torch:
  - a tap launches it,
  - another tap switches it on,
  - another one switches it off,
  - swiping to the right or left changes the brightness level of the torch
- silent mode:
  - a green bell symbol means silent mode is off (so the watch will vibrate when receiving a notification),
  - taping it enables it: the icon becomes grey, and the watch will not vibrate when receiving notifications)
- settings: access to InfiniTime settings

The following settings are available:



- Display timeout (in seconds)
- Wake up: how to wake up the watch
  - nothing selected means only the left button wakes up the watch
  - single tap: tap one time on the screen to wake up the watch
  - double tap: tap two times to wake it up
  - raise wrist: screen will wake up when you raise your wrist
- Time format: 12h or 24h
- Watch face: choose between digital, analog and PineTimeStyle
- Steps: define your daily goal
- Set date: allows manually setting date
- Set time: allows manually setting time
- Battery: displays battery level and voltage
- Chimes: Emit a small vibration every hour or half-hour
- Shake calibration: calibration the sensitivity of the “shake to wake” functionality
- Firmware: displays information about the InfiniTime version
- About: displays information about InfiniTime, the Bootloader, uptime, etc

2.2 Flash And Upgrade

2.2.1 Bootloader, Firmware and recovery firmware

Firmware, InfiniTime, Bootloader, Recovery firmware, OTA, DFU… What is it?

You may have already encountered these words by reading the announcement, release notes, or the wiki guide and you may find them confusing if you're not familiar with the project.

A **firmware** is software running on the embedded hardware of a device.

InfiniTime has three distinct firmwares:

- InfiniTime is the operating system.
- The bootloader is responsible for safely applying firmware updates and runs before booting into InfiniTime.
- **The recovery firmware** is a special *application firmware* than can be loaded by the bootloader on user request. This firmware can be useful in case of serious issue, when the main application firmware cannot perform an OTA update correctly.

OTA (**O**ver **T**he **A**ir) refers to updating of the firmware over BLE (Bluetooth Low Energy). This is a functionality that allows the user to update the firmware on their device wirelessly.

DFU (**D**evice **F**irmware **U**pdate) is the file format and protocol used to send the update of the firmware to the watch over-the-air. InfiniTime implements the (legacy) DFU protocol from Nordic Semiconductor (NRF).

Bootloader

The bootloader is run right before booting into InfiniTime.
It is easily recognizable with its white pine cone that is progressively drawn in green. It also displays its own version at the bottom (1.0.0 as of now).

(add link to https://github.com/InfiniTimeOrg/InfiniTime/blob/develop/doc/gettingStarted/bootloader-1.0.jpg)

Most of the time, the bootloader just runs without your intervention (update and load the firmware).

However, you can enable 2 functionalities using the push button:

- Push the button until the pine cone is drawn in blue to force the rollback of the previous version of the firmware, even if you've already validated the updated one
- Push the button until the pine cone is drawn in red to load the recovery firmware. This recovery firmware only provides BLE connectivity and OTA functionality.

More info about the bootloader on its project page.

The firmware

Well, it's InfiniTime :)

You can check the InfiniTime version by first swiping right on the watchface to open quick settings, tapping the cogwheel to open settings, swipe up until you find an entry named "About" and tap on it.

(add link to https://github.com/InfiniTimeOrg/InfiniTime/raw/develop/doc/gettingStarted/version-1.0.jpg)

Recovery firmware

The *recovery functionality* allows to load a recovery firmware from the external flash memory to recover the PineTime when the current firmware cannot boot anymore.

This recovery firmware is a slightly modified version of InfiniTime that only provides a basic UI and the OTA functionality. You'll be able to use this firmware to load a new firmware over-the-air using BLE connectivity.

This article describes how to upgrade your PineTime to benefit from this feature.

PineTime units shipped after (confirm date with JF) come with the recovery firmware already installed, so there's no need to follow this procedure.

2.2.2 Upgrading your PineTime

There are 2 ways to upgrade your PineTime:

- "Over-The-Air", i.e. using the Bluetooth connectivity to send firmware from a companion app: this is recommended for sealed devices
- using the SWD interface: only possible for dev / non-sealed units, as it requires access to the internals of the watch.

Over-The-Air (OTA)

To update your PineTime, you can use one of the compatible companion applications.

The updating process differs slightly on every companion app, so you'll need to familiarize yourself with the companion app of your choice.

All releases of InfiniTime are available on the release page of the GitHub repo under assets.

To update the firmware, you need to download the DFU of the firmware version that you'd like to install, for example `pinetime-mcuboot-app-dfu-1.6.0.zip`, and flash it with your companion app.

Using Gadgetbridge

(Pics from the original article will be added soon).

Connecting to Gadgetbridge

- Launch Gadgetbridge and tap on the "+" button on the bottom right to add a new device: (add pic)
- Wait for the scan to complete, your PineTime should be detected: (add pic)
- Tap on it. Gadgdetbridge will pair and connect to your device: (add pic)

Updating with Gadgetbridge

Now that Gadgetbridge is connected to your PineTime, use a file browser application and find the DFU file (`pinetime-mcuboot-app-dfu-x.x.x.zip`) you downloaded previously.
Tap on it and open it using the Gadgetbridge application/firmware installer:
(add pic)

Read carefully the warning and tap Install:
(add pic)

Wait for the transfer to finish. Your PineTime should reset and reboot with the new version of InfiniTime!

Don't forget to validate your firmware. In the InfiniTime go to the settings (swipe right, select gear icon) and Firmware option and click validate. Otherwise after reboot the previous firmware will be used.

(add pic)

Using Amazfish

Please see this article which describes how to use Amazfish on Sailfish OS to upgrade your PineTime.
Instructions also apply if you're running Amazfish on Linux.

Using ITD

ITD comes with a graphical user interface, called `itgui`, which allows upgrading InfiniTime.
Please see ITD's README for more details.

Using NRFConnect

- Open NRFConnect. Swipe down in the Scanner tab and wait for your device to appear:
  (add pic)
- Tap on the *Connect* button on the right of your device. NRFConnect will connect to your PineTime and discover its characteristics. Tap on the DFU button on the top right:
  (add pic)
- Select Distribution packet (ZIP):
  (add pic)
- Find the DFU file (`pinetime-mcuboot-app-dfu-x.x.x.zip`) you downloaded previously, the DFU transfer will start automatically. When the transfer is finished, your PineTime will reset and restart on the new version of InfiniTime! Don't forget to validate your firmware. In the InfiniTime go to the settings (swipe right, select gear icon) and Firmware option and click validate. Otherwise after reboot the previous firmware will be used.

Using the SWD interface

Download the files bootloader.bin, image-x.y.z.bin and pinetime-graphics-x.y.z.bin from the releases page.

The bootloader reads a boot logo from the external SPI flash memory. The first step consists of flashing a tool in the MCU that will flash the boot logo into this SPI flash memory. This first step is optional but recommended (the bootloader will display garbage on screen for a few second if you don't do it). Using your SWD tool, flash pinetime-graphics-x.y.z.bin at offset 0x0000. Reset the MCU and wait for a few second, until the logo is completely drawn on the display.

Then, using your SWD tool, flash those file at specific offset:

- bootloader.bin : **0x0000**
- image-x.y.z.bin : **0x8000**

Reset and voilà, you're running InfiniTime on your PineTime!

2.2.3 Firmware validation

Firmware updates must be manually validated. If the firmware isn't validated and the watch resets, the watch will revert to the previous firmware. This is a safety feature to prevent bricking your device with faulty firmware.

You can validate your updated firmware on InfiniTime >= 1.0 by following this simple procedure:

- From the watchface, swipe right to display the quick settings menu
- Open settings by tapping the cogwheel on the bottom right
- Swipe up until you find an entry named Firmware and tap on it
- If the firmware is not validated yet, you can either validate the running firmware, or reset and revert to the previous firmware version

2.3 Troubleshooting

2.3.1 Bluetooth connectivity

InfiniTime versions prior to 1.6.0 "Ice Apple" had a known BLE bug, which caused lost of Bluetooth connectivity after a few hours.
The only way to fix this was to restart the watch (by holding the button for ~8 seconds).

This bug has been fixed, and with 1.6 and onwards, BLE connectivity is a lot more reliable.

CHAPTER

THREE

HOW TO USE INFINITIME APPS

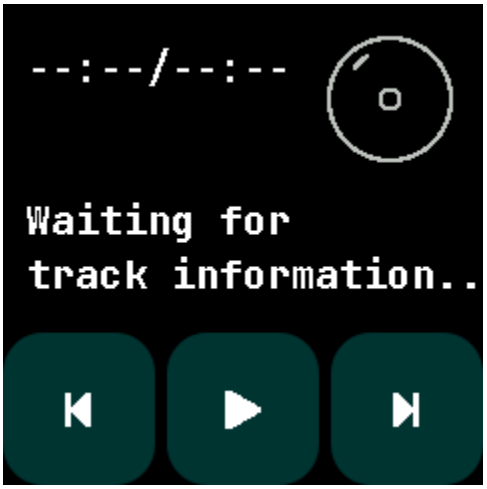
3.1 Stopwatch

The stopwatch measures passed time in minutes, seconds and hundredths of seconds.

When the stop watch is not running, the left button shows a stop symbol and resets the current time to zero. It is not clickable when the time is already zero. The right button shows a start symbol and starts the stop watch.

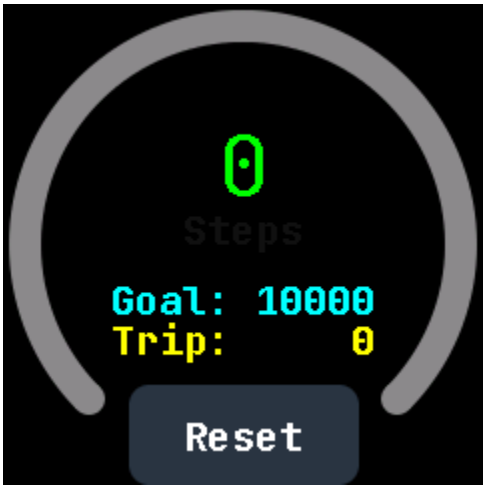
While the stop watch is running, the left button shows a flag. Tapping it allows to register an intermediate time without stopping the measurement. The last two intermediate times are shown below the running time besides. The right button shows a pause symbol. Tapping it pauses the measurement (without resetting it).

3.2 Music player



3.3 Navigation directions

3.4 Steps

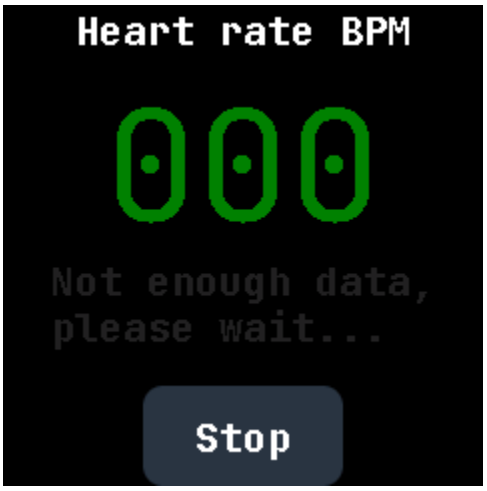


The steps screen shows the current number of steps in the middle and your current steps goal and trip count underneath. Around it is a circular meter that shows how much of your goal you already achieved.

The steps count is measured daily and will be reset at midnight. You can configure your step goal in the settings.

Your trip steps will not be reset at midnight. You can do this yourself by pressing the Reset button on the bottom of the screen.

3.5 Heart-rate



The heart rate app measures your heart rate using the sensor on the backside of the watch. When you activate the heart rate measurement, a fast blinking green light comes out of .

At the beginning, the measurement is turned off. The screen shows a grey “000” and “stopped” underneath. Press the “Start”-button at the bottom of the screen activate the measurement. After starting the measurement, the measurement should become green and the text underneath should be “Not enough data, please wait …”. After a short while (~ 10 seconds), the text should be “Measuring …” and on the top there should be a number that is your measured heart rate.

To stop the measurement, press the button at the bottom, which should now be labeled with “Stop”.

Keep in mind that InfiniTime and the PineTime watch are no medical devices. The measured number is not guaranteed to be your actual heart rate. See this feature more as a toy.

3.6 Timer

The Timer app lets you set a timer after which your watch will vibrate once and wake up if it was asleep.

The screen shows two digits for minutes and two digits for seconds, each with a button labeled "+" above and a button labeled "-" beneath. Tapping those buttons lets you set the desired duration until the timer vibrates.

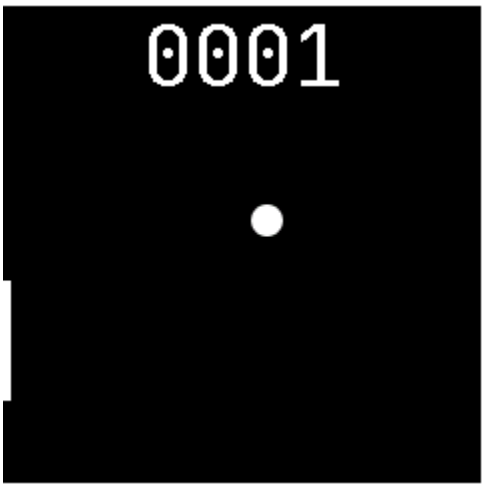
At the bottom of the screen is a button with a play icon. Tapping it activates the timer and changes the play button to a pause button. Tapping the pause button pauses the timer.

3.7 Paint

InfiniPaint is an app that lets you draw using the touchscreen.

At the start, the app will show a black screen. Touching the screen will let you draw white pixels. If you hold a touch for long enough, there will be a vibration and you will have a new color.

3.8 Pong

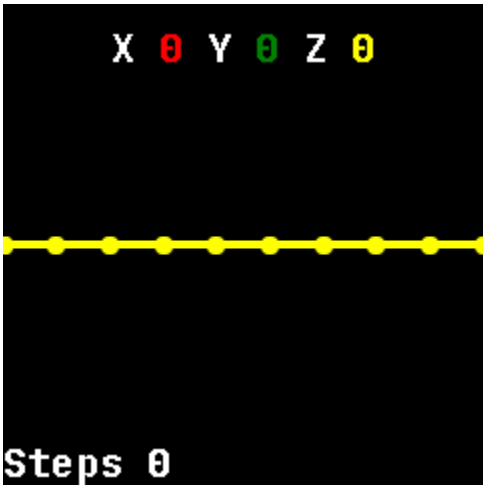


Pong is a little game that you can play. In the middle of the screen is a little ball that will bounce of the walls except the left one. At the left side there is a bar that will bounce the ball. You can control it by touching the screen Each time you keep the ball inside the screen will earn you one point. Your points are shown at the top of the screen. Try to get as many points as possible.

3.9 2048

2048 is a puzzle game you can play. At the start, you can see a four by four grid of cells and your score at the top, which should be zero. Some of the cells will be filled with numbers that are powers of two. Your goal is to make as many moves as possible and thereby earn as many points as possible. You make a move by swiping either up, down, left or right. All numbers on the grid will then move as far in this direction as possible. When two numbers with the same values collide, they will be added to a single cell.

3.10 Accelerometer



The accelerometer visualizes the accelerations along the x, y and z axis that are measured by the motion sensor of the watch. In the bottom left corner, it also shows your current step count.

3.11 Metronome

The metronome app lets you have a vibrating metronome on your wrist. The metronome vibrates on each beat. The first beat of each bar is stronger than the other ones.

Move the dot on the arc to change the bpm (beats per minute) that are shown in the middle.

On the bottom are two buttons. Tapping the left one opens a scrollable list from which you can select how many bpb (beats per bar) you would like to have. Tap a number in the list to select it. The right button shows a play sign if the metronome is paused and a pause sign if the metronome is running. Use it to start or pause the metronome.

3.12 Alarm

The alarm app lets you set an alarm to wake you up in the morning (or whenever).

The screen contains two digits for the hour on the left and two digits for the minutes on the right. Above and beneath each are buttons labeled with "+" and "-" that let you modify the time.

At the bottom of the screen there are two buttons. The left one is either grey and labeled "OFF" or green and labeled "ON". Tapping it, toggles it between those two. It shows whether the alarm is active or not. The right button is labeled either "ONCE", "DAILY" or "MON-FRI". Tapping it cycles between those three. It shows on which days the alarm will ring when it is active.

At the top, in the middle is a button labeled "i". Tapping it informs you, how many time is left before the next alarm.

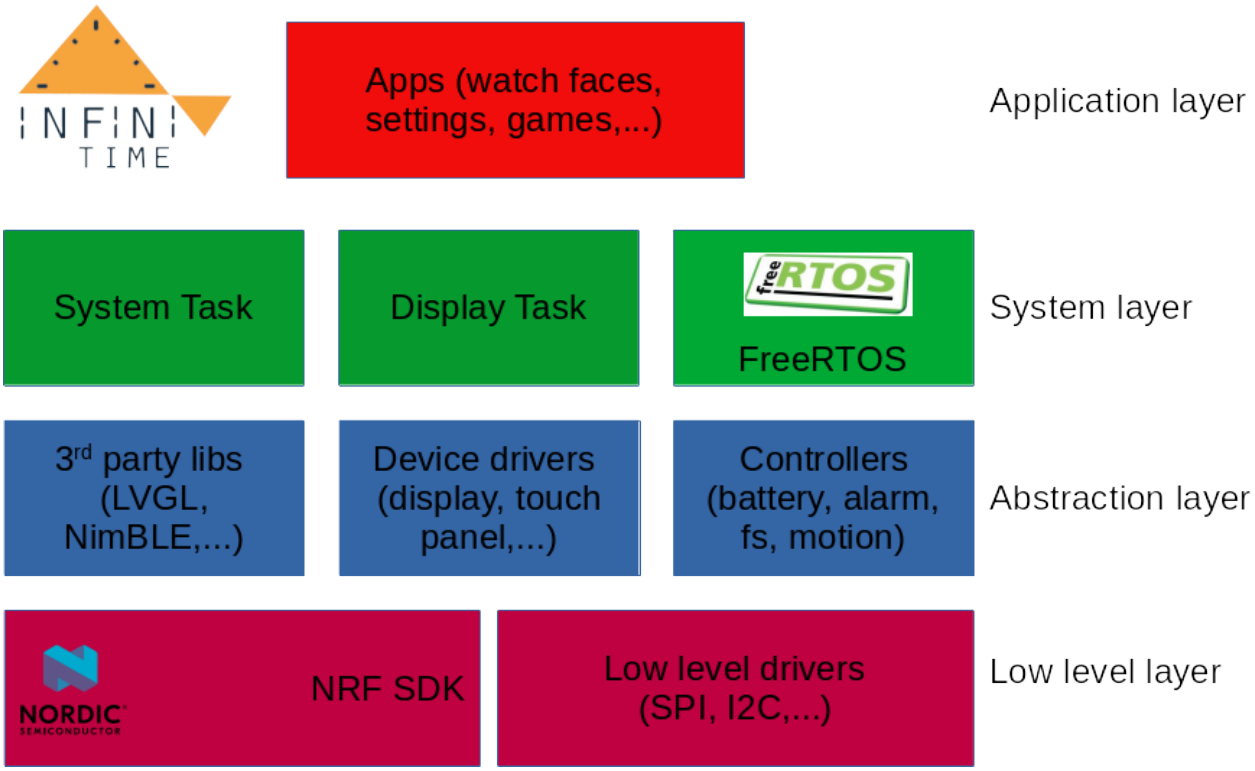
When the alarm rings, the watch vibrates repeatedly and shows the alarm app. The bottom left button is then red and labeled with a stop sign. Press it, to end the ringing.

CHAPTER

FOUR

DEVELOPER DOCUMENTATION

4.1 Global architecture



The project is composed of multiple layers:

4.1.1 The low level layer

The low level layer is composed of mostly hardware dependent software modules : startup code, low level drivers for clocks, busses, peripherals,… Most of these modules are provided by the NRF SDK, but some drivers (I2C and SPI, for example) were reimplemented to better fit the needs of the project.

4.1.2 The abtraction layer

The abstraction layer is composed of components and controllers that abtracts the hardware from the upper layers while providing higher level APIs and services for these layers.

Those controllers are, among others:

- The **battery controller** that handles the battery level measurements and conversion
- The **file system** that allows the application to easily read and write data to the flash memory
- The **motion controller** that provides motion and step information from the motion sensor
- and many others...

4.1.3 The system layer

The system layer is composed of the RTOS (FreeRTOS) and the tasks needed for the firmware to run properly. These tasks are mainly the System task and the display task. This layer also implements a message passing mecanism (based on message queues) that allows tasks to exchange messages and events.

FreeRTOS

InfiniTime is based on FreeRTOS, a real-time operating system. FreeRTOS provides several quality of life abstractions (for example easy software timers) and most importantly supports multiple tasks. If you want to read up on real-time operating systems, you can look here and here. The main "process" (the function `main()`)creates at least one task and then starts the FreeRTOS task scheduler. The task scheduler is responsible for giving every task enough cpu time. As there is only one core on the SoC of the PineTime, real concurrency is impossible and the scheduler has to swap tasks in and out to emulate it.

Tasks

Tasks are created by calling `xTaskCreate()` and passing a function with the signature `void functionName(void*)`. For more info on task creation see the FreeRTOS Documentation.

System task is the main task of the project. It's responsible for initializing all devices and peripheral at startup, for managing and routing events between the tasks and the controllers, for managing the wake and sleep mode, for responding to external events (button press, touch even),... It's implement in the class `PineTime::System::SystemTask`. It provides 2 main methods: `SystemTask::Start()` that starts the task, and `SystemTask::Work()` which is the method the task executes. It starts by initializing all device and peripherals and then enters into its main loop.

Display task is implemented in the class `PineTime::Applications::DisplayApp`. Similarly to **System task**, it provides the methods `Start()` and `Work()`. Display task handles the whole UI : it responds to touch and button events, it switches between apps, refresh the UI according to user interactions,...

The BLE stack (NimBLE) creates 2 other tasks (**ll** and **ble**) to handle the whole BLE processing.

An additional task handles the signal processing for the heart rate monitor : `PineTime::Applications::HeartRateTask`.

4.1.4 The application layer

The **application layer** implements the applications visible to the user (watch faces, settings, music application, games,...).

The UI is based on the *Light and versatile graphics library (LVGL)*.

4.2 Bluetooth

Header files with short documentation for the functions are inside libs/mynewt-nimble/nimble/host/include/host/.

4.3 How to implement an app

4.3.1 Theory

The user interface of InfiniTime is made up of **screens**. Screens that are opened from the app launcher are considered **apps**. Every app in InfiniTime is it's own class. An instance of the class is created when the app is launched and destroyed when the user exits the app. They run inside the "displayapp" task (briefly discussed here). Apps are responsible for everything drawn on the screen when they are running. By default, apps only do something (as in a function is executed) when they are created or when a touch event is detected.

4.3.2 Interface

Every app class has to be inside the namespace `Pinetime::Applications::Screens` and inherit from `Screen`. The constructor should have at least one parameter `DisplayApp* app`, which it needs for the constructor of its parent class Screen. Other parameters should be references to controllers that the app needs. A destructor is needed to clean up LVGL and restore any changes (for example re-enable sleeping). App classes can override `bool OnButtonPushed()`, `bool OnTouchEvent(TouchEvents event)` and `bool OnTouchEvent(uint16_t x, uint16_t y)` to implement their own functionality for those events. If an app only needs to display some text and do something upon a touch screen button press, it does not need to override any of these functions, as LVGL can also handle touch events for you. If you have any doubts, you can always look at how the other apps are doing things.

Continuous updating

If your app needs to be updated continuously, you can do so by overriding the `Refresh()` function in your class and calling `lv_task_create` inside the constructor.

An example call could look like this:

```
taskRefresh = lv_task_create(RefreshTaskCallback, LV_DISP_DEF_REFR_PERIOD, LV_TASK_PRIO_
↪MID, this);
```

With `taskRefresh` being a member variable of your class and of type `lv_task_t*`. Remember to delete the task again using `lv_task_del`. The function `RefreshTaskCallback` is inherited from `Screen` and just calls your `Refresh` function.

4.3.3 Creating your own app

A minimal app could look like this:

MyApp.h:

```
#pragma once

#include "displayapp/screens/Screen.h"
#include <lvgl/lvgl.h>

namespace Pinetime {
  namespace Applications {
    namespace Screens {
      class MyApp : public Screen {
      public:
        MyApp(DisplayApp* app);
        ~MyApp() override;
      };
    }
  }
}
```

MyApp.cpp:

```
#include "displayapp/screens/MyApp.h"
#include "displayapp/DisplayApp.h"

using namespace Pinetime::Applications::Screens;

MyApp::MyApp(DisplayApp* app) : Screen(app) {
  lv_obj_t* title = lv_label_create(lv_scr_act(), nullptr);
  lv_label_set_text_static(title, "My test application");
  lv_label_set_align(title, LV_LABEL_ALIGN_CENTER);
  lv_obj_align(title, lv_scr_act(), LV_ALIGN_CENTER, 0, 0);
}

MyApp::~MyApp() {
  lv_obj_clean(lv_scr_act());
}
```

Both of these files should be in displayapp/screens/ or displayapp/screens/settings/ if it's a setting app.

Now we have our very own app, but InfiniTime does not know about it yet. The first step is to include your MyApp.cpp (or any new cpp files for that matter) in the compilation by adding it to CMakeLists.txt. The next step to making it launchable is to give your app an id. To do this, add an entry in the enum class `Pinetime::Applications::Apps` (displayapp/Apps.h). Name this entry after your app. Add `#include "displayapp/screens/MyApp.h"` to the file displayapp/DisplayApp.cpp. Now, go to the function `DisplayApp::LoadApp` and add another case to the switch statement. The case will be the id you gave your app earlier. If your app needs any additional arguments, this is the place to pass them.

If you want to add your app in the app launcher, add your app in displayapp/screens/ApplicationList.cpp to one of the `CreateScreen` functions, or add another `CreateScreen` function if there are no empty spaces for your app. If your app is a setting, do the same procedure in displayapp/screens/settings/Settings.cpp.

You should now be able to build the firmware and flash it to your PineTime. Yay!

Please remember to pay attention to the UI guidelines when designing an app that you want to be included in InfiniTime.

4.4 Generating the fonts and symbols

You can download fonts using the links below:

- Jetbrains Mono
- Awesome font from LVGL
- Open Sans Light from Google

4.4.1 Generate the fonts:

- Open the LVGL font converter
- Name : jetbrains_mono_bold_20
- Size : 20
- Bpp : 1 bit-per-pixel
- Do not enable font compression and horizontal subpixel hinting
- Load the file `JetBrainsMono-Bold.tff` (use the file in this repo to ensure the version matches) and specify the following range : `0x20-0x7f, 0x410-0x44f`
- Add a 2nd font, load the file `FontAwesome5-Solid+Brands+Regular.woff` and specify the following range : `0xf293, 0xf294, 0xf244, 0xf240, 0xf242, 0xf243, 0xf241, 0xf54b, 0xf21e, 0xf1e6, 0xf54b, 0xf017, 0xf129, 0xf03a, 0xf185, 0xf560, 0xf001, 0xf3fd, 0xf069, 0xf1fc, 0xf45d, 0xf59f, 0xf5a0, 0xf029, 0xf027, 0xf028, 0xf6a9, 0xf04b, 0xf04c, 0xf048, 0xf051, 0xf095, 0xf3dd, 0xf04d, 0xf2f2, 0xf024, 0xf252, 0xf569, 0xf201, 0xf06e, 0xf015`
- Click on Convert, and download the file `jetbrains_mono_bold_20.c` and copy it in `src/DisplayApp/Fonts`
- Add the font .c file path to src/CMakeLists.txt
- Add an LV_FONT_DECLARE line in src/libs/lv_conf.h

Add new symbols:

- Browse the cheatsheet and find your new symbols
- For each symbol, add its hex code (0xf641 for the 'Ad' icon, for example) to the *Range* list (Remember to keep this readme updated with newest range list)
- Convert this hex value into a UTF-8 code using this site
- Define the new symbols in `src/displayapp/screens/Symbols.h`:

```
static constexpr const char* newSymbol = "\xEF\x86\x85";
```

4.4.2 Simple method to generate a font

If you want to generate a basic font containing only numbers and letters, you can use the above settings but instead of specifying a range, simply list the characters you need in the Symbols field and leave the range blank. This is the approach used for the PineTimeStyle watchface. This works well for fonts which will only be used to display numbers, but will fail if you try to add a colon or other punctuation.

- Open the LVGL font converter
- Name : open_sans_light
- Size : 150
- Bpp : 1 bit-per-pixel
- Do not enable font compression and horizontal subpixel hinting
- Load the file `open_sans_light.tff` (use the file in this repo to ensure the version matches) and specify the following symbols : `0123456789`
- Click on Convert, and download the file `open_sans_light.c` and copy it in `src/DisplayApp/Fonts`
- Add the font .c file path to src/CMakeLists.txt (search for jetbrains to find the appropriate location/format)
- Add an LV_FONT_DECLARE line in src/libs/lv_conf.h (as above)

Navigation font

To create the navigtion.ttf I use the web app icomoon. This app can import the svg files from the folder *src/displayapp/icons/navigation/unique* and create a ttf file the project for the site is *lv_font_navi_80.json* you can import it to add or remove icons

You can also use the online LVGL tool to create the .c

ttf file : navigation.ttf name : lv_font_navi_80 size : 80px Bpp : 2 bit-per-pixel range : 0xe900-0xe929

$lv_font_conv –font navigation.ttf -r '0xe900-0xe929' –size 80 –format lvgl –bpp 2 –no-prefilter -o lv_font_navi_80.c

I use the method above to create the other ttf

4.5 Creating a stopwatch in PineTime

This article from Pankaj Raghav describes in details how to create a stopwatch app in InfiniTime.

4.6 Tips on designing an app UI

- Align objects all the way to the edge or corner
- Buttons should generally be at least 50px high
- Buttons should generally be on the bottom edge
- Make interactable objects **big**
- When using a page indicator, leave 8px for it on the right side
  - It is acceptable to leave 8px on the left side as well to center the content
- Top bar takes at least 20px + padding

  - Top bar right icons move 8px to the left when using a page indicator

- A black background helps to hide the screen border, allowing the UI to look less cramped when utilizing the entire display area.

(add link to developer-documentation/ui/example.png)

4.7 BLE implementation and API

CHAPTER

FIVE

BUILD, FLASH AND DEBUG

5.1 Project branches

5.2 Versioning

5.3 Files included in the release notes

5.4 Build the project

5.4.1 Dependencies

To build this project, you'll need:

- A cross-compiler : ARM-GCC (9-2020-q2-update)
- The NRF52 SDK 15.3.0 : nRF-SDK v15.3.0
- The Python 3 modules `cbor`, `intelhex`, `click` and `cryptography` modules for the `mcuboot` tool (see requirements.txt)
- To keep the system clean, you can install python modules into a python virtual environment (`venv`)

```
python -m venv .venv
source .venv/bin/activate
python -m pip install wheel
python -m pip install -r tools/mcuboot/requirements.txt
```

- A reasonably recent version of CMake (I use 3.16.5)

Note that you do not need to install those dependencies on your system if you use Docker to build the project.

5.4.2 Using the command line

Build steps

Clone the repo

Clone the repo and update the submodules:

```
git clone https://github.com/InfiniTimeOrg/InfiniTime.git
cd InfiniTime
git submodule update --init
mkdir build
cd build
```

Generate the project with CMake

Run CMake to generate the project. Here's the default command line

```
cmake -DARM_NONE_EABI_TOOLCHAIN_PATH=/path/to/the/toolchain -DNRF5_SDK_PATH=/path/to/the/
↪sdk ..
```

You can specify a few more options:

- `-DCMAKE_BUILD_TYPE` : specify the build type (`Release` or `Debug`). The `Release` mode is selected by default. It enables various optimizations from the compiler to reduce the binary size and increase perfromances. `Debug` mode does not enable those optimizations. It makes the binary easier to debug but it'll run slower and the binary file will be bigger.
- `-DBUILD_DFU` : enables (`1`) or disables (`0`) the generation of the DFU file (needed for this OTA udate). Note that this option needs adafruit-nrfutil installed on your system.

Build

Run make to build the project:

```
make -j
```

This command will build all the targets of the project (see below). You can also specify a specific target you want to build:

```
make -j pinetime-mcuboot-app
```

Here's the list of CMake targets:

- pinetime-app
- pinetime-mcuboot-app
- pinetime-recovery
- pinetime-mcuboot-recovery
- pinetime-recovery-loader
- pinetime-mcuboot-recovery-loader

Binary files are generated in the `src/` subfolder.

These files are named with various extensions:

- **.bin** files are binary files corresponding to the firmware. These files are intended to be ran by the MCU of the PineTime.
- **.hex** files are binary files formatted in the Intel HEX format.
- **.map** files contains the memory mapping generated by the compiler. Useful to understand the memory usage and mapping of the firmware.
- **.out** files are intended to be used by a debugger (GDB) to debug the firmware running on the PineTime

You'll find various files named according to their respective `cmake target`:

- **pinetime-app** : The standalone version of the firmware. This firmware must be flashed at offset 0x00 and runs without the bootloader. You can use this binary to debug (using a SWD debugger) the firmware faster and more easily as it doesn't rely on the bootloader.
- **pinetime-mcuboot-app** : The firmware with support for the bootloader. The file `pinetime-mcuboot-app-x.y.z.bin` must be converted into an *MCUBoot* image for the bootloader to be able to run it. This image is named pinetime-mcuboot-app-image-x-y-z.bin and it can be flashed at offset **0x8000**. A DFU file for the OTA update is also generated (if the option BUILD_DFU was enabled during `cmake` generation) : pinetime-mcuboot-app-dfu-x.y.z.zip.
- **pinetime-recovery** is the recovery firmware (standalone).
- **pinetime-mcuboot-recovery** is the bootloader variant of the recovery firmware. The MCUBoot image (pinetime-mcuboot-recovery-image) and the DFU file (pinetime-mcboot-recovery-dfu) are also generated.
- **pinetime-recovery-loader** is a firmware that updates the bootloader of the PineTime. Use this firmware with caution as it erases and overwrites the bootloader of your PineTime. There's no fallback in case of issue during this operation. The MCUBoot image and DFU file are also generated.

Most of the time, you'll only need the file `pinetime-mcuboot-app-image` to update the firmware using a SWD debugger or the file `pinetime=-mcuboot-app-dfu` to update it over the air using a BLE device.

See below for more info on flashing and upgrading your firmware on your PineTime.

5.4.3 Using docker

A Docker image (Dockerfile) containing all the build environment is available for X86_64 and AMD64 architectures. These images make the build of the firmware and the generation of the DFU file for OTA quite easy, as well as preventing clashes with any other toolchains or development environments you may have installed.

Based on Ubuntu 18.04 with the following build dependencies:

- ARM GCC Toolchain
- nRF SDK
- MCUBoot
- adafruit-nrfutil

Run a container to build the project

The `infinitime-build` image contains all the dependencies you need. The default `CMD` will compile sources found in `/sources`, so you need only mount your code.

Before continuing, make sure you first build the image as indicated in the Build the image section, or check the Using the image from Docker Hub section if you prefer to use a pre-made image.

This example will build the firmware, generate the MCUBoot image and generate the DFU file. For cloning the repo, see these instructions. Outputs will be written to **<project_root>/build/output**:

```
cd <project_root> # e.g. cd ./work/Pinetime
docker run --rm -it -v $(pwd):/sources infinitime-build
```

If you only want to build a single CMake target, you can pass it in as the first parameter to the build script. This means calling the script explicitly as it will override the `CMD`. Here's an example For `pinetime-app`:

```
docker run --rm -it -v $(pwd):/sources infinitime-build /opt/build.sh pinetime-app
```

The image is built using 1000:1000 for the user id and group id. If this is different to your user or group ids (run `id -u` and `id -g` to find out what your id values are if you are unsure), you will need to override them via the `--user` parameter in order to prevent permission errors with the output files (and the cmake build cache).

Running with this image is the same as above, you just specify the ids to `docker run`:

```
docker run --rm -it -v $(pwd):/sources --user $(id -u):$(id -g) infinitime-build
```

Or you can specify your user id and group id (by number, not by name) directly:

```
docker run --rm -it -v $(pwd):/sources --user 1234:1234 infinitime-build
```

Using the image from Docker Hub

NOTE : This image is provided by a contributor and might not be up-to-date.

The image is available via Docker Hub for both the amd64 and arm64v8 architectures at pfeerick/infinitime-build.

It can be pulled (downloaded) using the following command:

```
docker pull pfeerick/infinitime-build
```

The default `latest` tag *should* automatically identify the correct image architecture, but if for some reason Docker does not, you can specify it manually:

- For AMD64 (x86_64) systems: `docker pull pfeerick/infinitime-build:amd64`
- For ARM64v8 (ARM64/aarch64) systems: `docker pull pfeerick/infinitime-build:arm64v8`

Build the image

You can build the image yourself if you like!

The following commands must be run from the root of the project. This operation will take some time but, when done, a new image named *infinitime-build* is available.

```
docker image build -t infinitime-build ./docker
```

The `PUID` and `PGID` build arguments are used to set the user and group ids used in the container, meaning you will not need to specify it later unless they change for some reason. Specifying them is not mandatory, as this can be over-ridden at build time via the `--user` flag, but doing so will make the command you need to run later a bit shorter. In the below examples, they are set to your current user id and group id automatically. You can specify them manually, but they must be specified by number, not by name.

```
docker image build -t infinitime-build --build-arg PUID=$(id -u) --build-arg PGID=$(id -
↪g) ./docker
```

5.4.4 Using VSCode

The .VS Code folder contains configuration files for developing InfiniTime with VS Code. Effort was made to have these rely on Environment variables instead of hardcoded paths.

Environment Setup

To support as many setups as possible the VS Code configuration files expect there to be certain environment variables to be set.

Variable	Description	Example
ARM_NONE_EABI_TOOLCHAIN_PATH	path to the toolchain directory	`export ARM_NONE_EABI_TOOLCHAIN_PATH=/opt/gcc-arm-none-eabi-9-2020-q2-update`
NRF5_SDK_PATH	path to the NRF52 SDK	`export NRF5_SDK_PATH=/opt/nRF5_SDK_15.3.0_59ac345`

VS Code Extensions

We leverage a few VS Code extensions for ease of development.

Required Extensions

- C/C++ - C/C++ IntelliSense, debugging, and code browsing.
- CMake Tools - Extended CMake support in Visual Studio Code

Optional Extensions

Cortex-Debug - ARM Cortex-M GDB Debugger support for VS Code

Cortex-Debug is only required for interactive debugging using VS Codes built in GDB support.

VS Code/Docker DevContainer

The .devcontainer folder contains the configuration and scripts for using a Docker dev container for building InfiniTime

Using the Remote-Containers extension is recommended. It will handle configuring the Docker virtual machine and setting everything up.

More documentation is available in the readme in .devcontainer

DevContainer on Ubuntu

To use the DevContainer configuration on Ubuntu based systems two changes need to be made:

1. Modify the file `.devcontainer/devcontainer.json` and add the argument `"--net=host"` to the `"runArgs"` parameter making the line look like this: `"runArgs": [ "--cap-add=SYS_PTRACE", "--security-opt", "seccomp=unconfined", "--net=host"],`
2. Modify the file `.vscode/launch.json` and change the argument of `"gdbTarget"` to `"127.0.0.1:3333"`, making the line look like: `"gdbTarget": "127.0.0.1:3333",`
3. To start debugging launch openocd on your host system with the appropriate configuration, for example with a stlink-v2 the command is: `openocd -f interface/stlink.cfg -f target/nrf52.cfg`. This launches openocd with the default ports `3333`, `4444` and `6666`.
4. In VsCode go to the Debug pane on the left of the screen and select the configuration `Debug - Openocd docker Remote` and hit the play button on the left.

5.5 Build the documentation

5.5.1 Setup

The documentation is written in Markdown (.md) files and generated using the Sphinx documentation generator.

First, we need to install Sphinx and its dependencies:

- `myst-parser` : add support for markdown files
- `sphinx_rtd_theme` : theme from ReadTheDocs

```
pip install sphinx
pip install myst-parser
pip install sphinx_rtd_theme
```

5.5.2 Build the doc

Run the following command in the folder `docs`

```
sphinx-build -b html ./ ./generated
```

Then display the doc by browsing to `generated/index.html` using your favorite web browser.

5.6 Flash the firmware using OpenOCD and STLinkV2

5.7 Build the project with Docker

5.8 Build the project with VSCode

5.9 Bootloader, OTA and DFU

5.10 Stub using NRF52-DK

5.11 Logging with JLink RTT

5.12 Using files from the releases

CHAPTER

SIX

HOW TO CONTRIBUTE

6.1 Report bugs

Have you found a bug in the firmware? Create an issue on Github explaining the bug, how to reproduce it, the version of the firmware you use…

6.2 Write and improve documentation

Documentation might be incomplete, or not clear enough, and it is always possible to improve it with better wording, pictures, photo, video,…

As the documentation is part of the source code, you can submit your improvements to the documentation by submitting a pull request (see below).

6.3 Fix bugs, add functionalities and improve the code

You want to fix a bug, add a cool new functionality or improve the code? See *How to submit a pull request below*.

CHAPTER

SEVEN

HOW TO SUBMIT A PULL REQUEST?

7.1 TL;DR

- Create a branch from develop
- Work on a single subject in this branch. Create multiple branches/pulls-requests if you want to work on multiple subjects (bugs, features,...)
- Test your modifications on the actual hardware
- Check your code against the coding conventions and clang-format and clang-tidy
- Clean your code and remove files that are not needed
- Write documentation related to your new feature if applicable
- Create a pull request and write a great description about it: what does your PR do, why, how,... Add pictures and video if possible
- Wait for someone to review your PR and take part in the review process
- Your PR will eventually be merged :)

Your contributions are more than welcome!

If you want to fix a bug, add functionality or improve the code, you'll first need to create a branch from the **develop** branch (see this page about the branching model). This branch is called a feature branch, and you should choose a name that explains what you are working on (ex: "add-doc-about-contributions"). In this branch, **focus on only one topic, bug or feature**. For example, if you created this branch to work on the UI of a specific application, do not commit modifications about the SPI driver. If you want to work on multiple topics, create one branch for each topic.

When your feature branch is ready, **make sure it actually works** and **do not forget to write documentation** about it if it's relevant.

Creating a pull request containing modifications that haven't been tested is strongly discouraged. If for any reason you cannot test your modifications, but want to publish them anyway, **please mention it in the description**. This way, other contributors might be willing to test it and provide feedback about your code.

Before submitting a PR, check your code against the coding conventions. This project also provides clang-format and clang-tidy configuration files. You should use them to ensure correct formatting of your code.

Don't forget to check the files you are going to commit and remove those which aren't necessary (config files from your IDE, for example). Remove old comments, commented code,...

Then, you can submit a pull request for review. Try to **describe your pull request as much as possible**: what did you do in this branch, how does it work, how it is designed, are there any limitations,... This will help the contributors to understand and review your code easily. You can add pictures and video to the description so that contributors will have a quick overview of your work.

Other contributors can post comments about the pull request, maybe ask for more info or adjustments in the code.

Once the pull request is reviewed and accepted, it'll be merged into **develop** and will be released in the next version of the firmware.

7.2 Why all these rules?

Reviewing pull requests is a **very time consuming task**. Everything you do to make reviewing easier will **get your PR merged faster**.

Reviewers will first look at the **description**. If it's easy to understand what the PR does, why the modification is needed or interesting and how it's done, a good part of the work is already done : we understand the PR and its context.

Reviewing **a few files that were modified for a single purpose** is a lot easier than reviewing 30 files modified for many reasons (bug fix, UI improvements, typos in doc,...), even if all the changes make sense. Also, it's possible that we agree on some modification but not on another, so we won't be able to merge the PR because of the changes that are not accepted.

The code base should be kept as consistent as possible. If the formatting of your code is not consistent with the rest of the code base, we'll ask you to review it.

Lastly the changes are tested. If it doesn't work out of the box, we'll ask you to review your code and to ensure that it works as expected.

It's totally normal for a PR to need some more work even after it was created, that's why we review them. But every round trip takes time, so it's good practice to try to reduce them as much as possible by following those simple rules.

CHAPTER

EIGHT

GOING FURTHER

- The PineTime wiki
- InfiniTime resources from other people
  - Videos
  - Articles

CHAPTER

NINE

LICENCES

This project is released under the GNU General Public License version 3 or, at your option, any later version.

It integrates the following projects:

- RTOS : **FreeRTOS** under the MIT license
- UI : **LittleVGL/LVGL** under the MIT license
- BLE stack : **NimBLE** under the Apache 2.0 license
- Font : **Jetbrains Mono** under the Apache 2.0 license

CHAPTER

TEN

CREDITS

Many people work on InfiniTime, creating PR, submitting bugs and issues, … There is also the whole #Pinetime community : a lot of people all around the world who are hacking, searching, experimenting and programming the Pinetime. We exchange our ideas, experiments and code in the chat rooms and forums.

Here are some people we would like to highlight:

- Atc1441 : He works on an Arduino based firmware for the Pinetime and many other smartwatches based on similar hardware. He was of great help when JF was implementing support for the BMA421 motion sensor and I^2C driver.
- Koen : He's working on a firmware based on RiotOS. He integrated similar libs as me : NimBLE, LittleVGL,… His help was invaluable too!
- Lup Yuen Lee : He is everywhere: he works on a Rust firmware, builds a MCUBoot based bootloader for the Pinetime, designs a Flutter based companion app for smartphones and writes a lot of articles about the Pinetime!

If you feel like you should appear on this list, just get in touch with us or submit a PR :)