

---

# **open source watch Documentation**

***Release 1.0.0***

**jj**

**Nov 24, 2019**



# CONTENTS

<b>1</b>	<b>About</b>	<b>3</b>
1.1	Todo . . . . .	3
1.2	Fast track . . . . .	3
<b>2</b>	<b>Install zephyr</b>	<b>5</b>
<b>3</b>	<b>zephyr on the pinetime smartwatch</b>	<b>7</b>
3.1	Blinky example . . . . .	7
<b>4</b>	<b>bluetooth (BLE) example</b>	<b>9</b>
4.1	Eddy Stone . . . . .	9
4.2	Ble Peripheral . . . . .	9
4.3	using Python to read out bluetoothservices . . . . .	10
<b>5</b>	<b>st7789 display</b>	<b>11</b>
5.1	Display example . . . . .	11
<b>6</b>	<b>Menuconfig</b>	<b>13</b>
6.1	Zephyr is like linux . . . . .	13
<b>7</b>	<b>hacking the pinetime smartwatch</b>	<b>15</b>
<b>8</b>	<b>Howto flash your zephyr image</b>	<b>17</b>
<b>9</b>	<b>howto generate pdf documents</b>	<b>19</b>



this document describes the installation of zephyr RTOS on the PineTime smartwatch.

<https://wiki.pine64.org/index.php/PineTime>

It should be applicable on other nordic nrf52832 based watches (Desay D6....).

the approach **in** this manual **is** to get quick results :

- minimal effort install
- **try** out the samples
- inspire you to modify **and** enhance

**suggestion :**

- install zephyr, <https://docs.zephyrproject.org>
- copy the board definition
- try some examples
- try out bluetooth
- try out the display



## ABOUT

I got a pinetime development kit very early.

I would like to thank the folks from <https://www.pine64.org/>.

I like to hack stuff, and I like the idea behind Open Source.

The smartwatches I hacked, contained microcontrollers from Nordic Semiconductor.

A lot of resources exist for this breed.

It is an Arm based, 32bit microcontroller with a lot of flash and RAM memory.

In fact it is a small computer on your wrist, with a battery and screen, and capable of bluetooth 4+ wireless communication.

A word of warning: this **is** work **in** progress.  
You're likely to have a better skillset than me.  
You are invited to add the missing pieces **and** to improve what's already there.

## 1.1 Todo

list with suggestions:

- better graphics
- watchdog
- DFU (update over bluetooth)
- acceleration sensor
- heart rate sensor
- fun stuff
- useless stuff, but somehow cool
- applications, e.g. calculator, cycle computer, step counter, heart attack predictor ...

## 1.2 Fast track

In this repository you can find modified directories, which you can copy to the zephyrproject directory:

- pinetime (board definition -> boards/arm)
- st7789v (example -> samples/display)

- blinky (example -> samples/basic)



## INSTALL ZEPHYR

[https://docs.zephyrproject.org/latest/getting\\_started/index.html](https://docs.zephyrproject.org/latest/getting_started/index.html)

the documentation describes an installation process under Ubuntu/macOS/Windows

I picked Debian (which is not listed) . . . and soon afterwards ran into trouble

*this behaviour is known as : stubborn or stupid, but I remain convinced it could work*

But even after following the rules, I got a problem with the `dtc` (device tree compiler)

- I solved this by creating a link from the development-tools to `/usr/bin/dtc` (here you need to make sure you got a very recent one)

```
cd /root/zephyr-sdk-0.10.3/sysroots/x86_64-pokysdk-linux/usr/bin/  
mv dtc dtc-orig  
ln -s /usr/bin/dtc dtc
```

**Note :** in order to get the display `st7789` Picture-Perfect, you might need a zephyr patch

have a look at : <https://github.com/zephyrproject-rtos/zephyr/pull/20570/files> You will find them in this repo under `patches-zephyr`.



## ZEPHYR ON THE PINETIME SMARTWATCH

### 3.1 Blinky example

**Note:** I think you need to connect the 5V, just connecting the SWD cable (3.3V) is likely not enough to light up the leds

The watch does **not** contain a led **as** such, but it has background leds **for** the LCD.  
Once lit, you can barely see it, cause the screen **is** black.

```
copy the board definition for the pinetime to the zephyrproject directory
$ cp (this repo)pinetime ~/zephyrproject/zephyr/boards/arm/pinetime

replace the blinky sample with the one in this repo
$ cp (this repo)blinky ~/zephyrproject/zephyr/samples/basic
```

have a look at the pinetime.dts file, here you see the definition of the background leds.

```
gpios = <&gpio0 14 GPIO_INT_ACTIVE_LOW>;
gpios = <&gpio0 22 GPIO_INT_ACTIVE_LOW>;
gpios = <&gpio0 23 GPIO_INT_ACTIVE_LOW>;
```

*building an image, which can be found under the build directory*

```
$ west build -p -b pinetime samples/basic/blinky
```

once the compilation is completed you can upload the firmware ~/zephyrproject/zephyr/build/zephyr/zephyr.bin



## BLUETOOTH (BLE) EXAMPLE

### 4.1 Eddy Stone

**Note:** compile the provided example, so a build directory gets created

```
$ west build -p -b pinetime samples/bluetooth/eddystone
```

this builds an image, which can be found under the build directory

I use linux with a bluetoothadapter 4.0. You need bluez.

```
#bluetoothctl  
[bluetooth]#scan on
```

And your Eddy Stone should be visible.

If you have a smartphone, you can download the nrf utilities app from nordic.

### 4.2 Ble Peripheral

this example is a demo of the services under bluetooth

first build the image

```
$ west build -p -b pinetime samples/bluetooth/peripheral -D CONF_FILE="prj.conf"
```

the image, can be found under the build directory, and has to be flashed to the pinetime

with linux you can have a look using bluetoothctl

```
#bluetoothctl  
[bluetooth]#scan on  
  
[NEW] Device 60:7C:9E:92:50:C1 Zephyr Peripheral Sample Long  
once you see your device  
[blueooth]#connect 60:7C:9E:92:50:C1 (the device mac address as displayed)  
  
then you can already see the services
```

same thing with the app from nordic, you could try to connect and display value of e.g. heart rate

## 4.3 using Python to read out bluetoothservices

In this repo you will find a python script : readbat.py In order to use it you need bluez on linux and the python *bluepy* module.

It can be used in conjunction with the peripheral bluetooth demo. It just reads out the battery level, and prints it.

```
import binascii
from bluepy.btle import UUID, Peripheral

temp_uuid = UUID(0x2A19)

p = Peripheral("60:7C:9E:92:50:C1", "random")

try:
    ch = p.getCharacteristics(uuid=temp_uuid)[0]
    print binascii.b2a_hex(ch.read())
finally:
    p.disconnect()
```

## ST7789 DISPLAY

### 5.1 Display example

**Note:** I think you need to connect the 5V, just connecting the SWD cable (3.3V) is likely not enough to light up the leds

The watch has background leds **for** the LCD.

They need to be on (LOW) to visualize the display.

```
replace the display sample with the one in this repo
$ cp (this repo)st7789 ~/zephyrproject/zephyr/samples/display
```

*building an image, which can be found under the build directory* the pinetime.overlay contains the definitions for the screen like resolution, and SPI-port data

```
$ west build -p -b pinetime samples/display/st7789v -DDTC_OVERLAY="pinetime.overlay"
```

once the compilation is completed you can upload the firmware ~/zephyrproject/zephyr/build/zephyr/zephyr.bin  
if all goes well, you should see some coloured squares on your screen





## MENUCONFIG

### 6.1 Zephyr is like linux

**Note:** to get a feel, compile a program, for example

```
west build -p -b pinetime samples/bluetooth/peripheral -D CONF_FILE="prj.conf"
```

the pinetime contains an external 32Kz crystal now you can have a look in the configuration file (and modify if needed)

```
$ west build -t menuconfig
```

```
Modules --->
Board Selection (nRF52832-MDK) --->
Board Options --->
SoC/CPU/Configuration Selection (Nordic Semiconductor nRF52 series MCU) --->
Hardware Configuration --->
ARM Options --->
Architecture (ARM architecture) --->
General Architecture Options --->
[ ] Floating point ----
General Kernel Options --->
Device Drivers ---> *****SELECT THIS ONE*****
C Library --->
Additional libraries --->
[*] Bluetooth --->
[ ] Console subsystem/support routines [EXPERIMENTAL] ----
[ ] C++ support for the application ----
System Monitoring Options --->
Debugging Options --->
[ ] Disk Interface ----
File Systems --->
-- Logging --->
Management --->
Networking --->
```

```
[ ] IEEE 802.15.4 drivers options ----
(UART_0) Device Name of UART Device for UART Console
[*] Console drivers --->
[ ] Net loopback driver ----
[*] Serial Drivers --->
Interrupt Controllers --->
Timer Drivers --->
```

(continues on next page)

(continued from previous page)

[illegible]

```
[*] NRF Clock controller support ---> <<<<<<<<<<<<<<<<<<<SELECT THIS ONE<<<<<<<<<
```

## HACKING THE PINETIME SMARTWATCH

The pinetime **is** preloaded **with** firmware.  
This firmware **is** secured, you cannot peek into it.

**Note:** the pinetime has a swd interface to write firmware you need special hardware I use a stm-link which is very cheap(2\$) You can also use the GPIO header of a raspberry pi / or orange pi (see my repo: <https://github.com/najnesnaj/openocd>)

To flash the software I use openocd : example for stm-link usb-stick

```
# openocd -s /usr/local/share/openocd/scripts -f interface/stlink.cfg -f target/nrf52.  
↪cfg
```

example for the orange-pi GPIO header (or raspberry)

```
# openocd -f /usr/local/share/openocd/scripts/interface/sysfsgpio-raspberrypi.cfg -c 'transport select swd'  
-f /usr/local/share/openocd/scripts/target/nrf52.cfg -c 'bindto 0.0.0.0'
```

once you started the openocd background server, you can connect to it using:

```
#telnet 127.0.0.1 4444
```

### programming

```
once your telnet sessions started:  
Trying 127.0.0.1...  
Connected to 127.0.0.1.  
Escape character is '^]'.  
Open On-Chip Debugger  
> program zephyr.bin  
  
target halted due to debug-request, current mode: Thread  
xPSR: 0x01000000 pc: 0x00001534 msp: 0x20004a10  
** Programming Started **  
auto erase enabled  
using fast async flash loader. This is currently supported  
only with ST-Link and CMSIS-DAP. If you have issues, add  
"set WORKAREASIZE 0" before sourcing nrf51.cfg/nrf52.cfg to disable it  
target halted due to breakpoint, current mode: Thread  
xPSR: 0x61000000 pc: 0x2000001e msp: 0x20004a10  
wrote 24576 bytes from file zephyr.bin in 1.703540s (14.088 KiB/s)  
** Programming Finished **
```

(continues on next page)

(continued from previous page)

```
And finally execute a reset :  
>reset
```

removing write protection see: [\*Howto flash your zephyr image\*](#)

## HOWTO FLASH YOUR ZEPHYR IMAGE

Once you completed your `west build`, your image is located under the build directory

```
$ cd ~/zephyrproject/zephyr/build/zephyr
here you can find zephyr.bin which you can flash
```

I have an orange pi (single board computer) in my network.

I copy the image using `$scp -P 8888 zephyr.bin 192.168.0.77:/usr/src/pinetime` (secure copy using my user defined port 8888 which is normally port 22)

---

**Note:** the PineTime watch is read/write protected executing the following : `nrf52.dap apreg 1 0x0c` shows 0x0

Mind you st-link does not allow you to execute that command, you need J-link. There is a workaround using the GPIO of a raspberry pi or a Orangepi. You have to reconfigure Openocd with the `-enable-cmsis-dap` option.

Unlock the chip by executing the command: `> nrf52.dap apreg 1 0x04 0x01`

---



## HOWTO GENERATE PDF DOCUMENTS

sphinx cannot generate pdf directly, and needs latex

```
apt-get install latexmk
apt-get install texlive-fonts-recommended
apt-get install xzdec
apt-get install cmap
apt-get install texlive-latex-recommended
apt-get install texlive-latex-extra
```