open source watch Documentation

Release 1.0.0

jj

Dec 11, 2019

CONTENTS

1 Install zephyr **3**

2 zephyr on the pinetime smartwatch **5**
2.1 Blinky example . . . 5

3 Reading out the button on the watch **7**
3.1 Building and Running . . . 7

4 bluetooth (BLE) example **9**
4.1 Eddy Stone . . . 9
4.2 Ble Peripheral . . . 9
4.3 using Python to read out bluetoothservices . . . 10

5 st7789 display **11**
5.1 Display example . . . 11

6 LittlevGL Basic Sample **13**
6.1 Overview . . . 13
6.2 Requirements . . . 13
6.3 Building and Running . . . 13
6.4 Todo . . . 13
6.5 References . . . 14

7 LittlevGL Clock Sample **15**
7.1 Overview . . . 15
7.2 Requirements . . . 15
7.3 Building and Running . . . 15
7.4 Todo . . . 16
7.5 References . . . 16

8 placing a button on the screen **17**
8.1 Building and Running . . . 17

9 Real Time Clock **19**
9.1 Overview . . . 19
9.2 Requirements . . . 19
9.3 Building and Running . . . 19
9.4 Todo . . . 19
9.5 References . . . 19

10 Serial Nor Flash **21**

10.1 Overview . . . 21
10.2 Requirements . . . 21
10.3 Building and Running . . . 22
10.4 Todo . . . 22
10.5 References . . . 22

11 sensors on the I2C bus **23**

12 configuring I2C **25**
12.1 board level definitions . . . 25
12.2 development trajectory . . . 25
12.3 defining an I2C sensor . . . 25
12.4 compiling the sample . . . 26

13 Bosch BMA280 **27**
13.1 Overview . . . 27
13.2 Requirements . . . 27
13.3 Building and Running . . . 27
13.4 Todo . . . 27
13.5 References . . . 28

14 Touchscreen Hynitron **29**
14.1 Overview . . . 29
14.2 Requirements . . . 29
14.3 Building and Running . . . 29
14.4 Todo . . . 29
14.5 References . . . 29

15 Menuconfig **31**
15.1 Zephyr is like linux . . . 31

16 hacking the pinetime smartwatch **33**

17 debugging the pinetime smartwatch **35**

18 scanning the I2C_1 port **37**
18.1 Building and Running . . . 37

19 howto flash your zephyr image **39**

20 howto generate pdf documents **41**

21 About **43**
21.1 Todo . . . 43
21.2 Fast track . . . 43


```
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

CHAPTER

ONE

INSTALL ZEPHYR

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 : stuborn 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.

CHAPTER

TWO

ZEPHYR ON THE PINETIME SMARTWATCH

2.1 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

CHAPTER

THREE

READING OUT THE BUTTON ON THE WATCH

```
The pinetime does have a button.
I do not have a segger debugging probe.
A way around this, it to put a value in memory at a fixed location.
With openocd you can peek at this memory location.
```

3.1 Building and Running

In this repo under samples you will find an adapted basic/button program. (copy this to the samples/basic directory)

```
west build -p -b pinetime samples/basic/button
```

Note: #define MY_REGISTER (*(volatile uint8_t*)0x2000F000)

in the program you can set values: MY_REGISTER=(read button value);

this way you know till whether the code executes

---

a way to set port 15 high (hard-coded of course :))

```
gpio_pin_configure(gpiob, 15,GPIO_DIR_OUT); //push button out
gpio_pin_write(gpiob, 15, &button_out); //set port high
```

```
#telnet 127.0.0.1 4444
```

Peeking

```
once your telnet sessions started:
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Open On-Chip Debugger
>mdw 0x2000F000 0x1
0x2000f000: 00000100 (switch pushed)
```

Note::

The watch has a button out port (15) and buttin in port (13). You have to set the out-port high. Took me a while to figure this out...

CHAPTER

FOUR

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()
```

CHAPTER

FIVE

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 While connecting 5V, do not connect 3.3V

```
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

```
$  west build -p -b pinetime samples/display/st7789v
```

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

CHAPTER

SIX

LITTLEVGL BASIC SAMPLE

6.1 Overview

This sample application displays “Hello World” in the center of the screen and a counter at the bottom which increments every second.

LittlevGL is a free and open-source graphics library providing everything you need to create embedded GUI with easy-to-use graphical elements, beautiful visual effects and low memory footprint.

6.2 Requirements

definitions can be found under the boards sub-directory

- pinetime.conf
- pinetime.overlay

The program has been modified to light up the background leds. Might be unnecessary... can be found in this repo

```
Matching labels are necessary!
pinetime.conf:CONFIG_LVGL_DISPLAY_DEV_NAME="DISPLAY"
pinetime.overlay:               label = "DISPLAY"; (spi definition)
```

6.3 Building and Running

Make sure you copied the board definitions.

```
west build -p -b pinetime samples/gui/lvgl
```

modifying the font size :

west build -t menuconfig goto additional libraries / lvgl gui library (look for fonts, and adapt according to your need)

west build

6.4 Todo

- Create a button
- touchscreen activation (problem cause zephyr does not support this yet)

- lvgl supports lv_canvas_rotate(canvas, &imd_dsc, angle, x, y, pivot_x, pivot_y) should be cool for a clock, chrono…

6.5 References

https://docs.littlevgl.com/en/html/index.html

LittlevGL Web Page: https://littlevgl.com/

CHAPTER

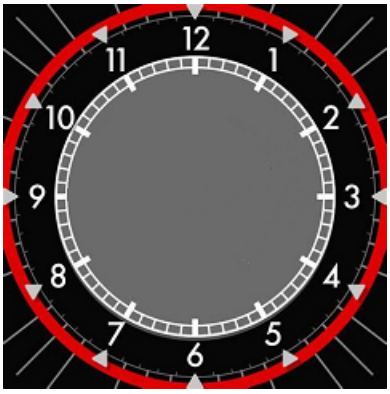
SEVEN

LITTLEVGL CLOCK SAMPLE

7.1 Overview

This sample application displays a “clockbackground” in the center of the screen.

LittlevGL is a free and open-source graphics library providing everything you need to create embedded GUI with easy-to-use graphical elements, beautiful visual effects and low memory footprint.



7.2 Requirements

Make sure the prj.conf contains the following :

```
CONFIG_LVGL=y
CONFIG_LVGL_OBJ_IMAGE=y
```

LitlevGL uses a “c” file to store the image. You need to convert a jpg, or png image to this c file. There is an online tool : https://littlevgl.com/image-to-c-array

7.3 Building and Running

copy the samples/gui/clock from this repository to the zephyr one.

```
west build -p -b pinetime samples/gui/clock
```

7.4 Todo

- create an internal clock (and adjustment mechanism, eg. bluetooth cts)
- lvgl supports lv_canvas_rotate(canvas, &imd_dsc, angle, x, y, pivot_x, pivot_y) should be cool for a clock, chrono...

7.5 References

https://docs.littlevgl.com/en/html/index.html

LittlevGL Web Page: https://littlevgl.com/

CHAPTER

EIGHT

PLACING A BUTTON ON THE SCREEN

8.1 Building and Running

In this repo under samples you will find an adapted gui/clock program. A button from the LVGL library is placed on the screen.

Later on when the touch-screen driver is ready, we'll be able to manipulate it.

```
Make sure that prj.conf file in clock directory contains the following:
```

Note: CONFIG_LVGL_OBJ_CONTAINER=y CONFIG_LVGL_OBJ_BUTTON=y

```
```

CHAPTER

NINE

REAL TIME CLOCK

9.1 Overview

This sample application “clock” uses the RTC0 timer. It uses the counter driver.

9.2 Requirements

Make sure the prj.conf contains the following :

```
CONFIG_COUNTER=y
```

You need the Kconfig file, which contains :

```
config COUNTER_RTC0
        bool
        default y if SOC_FAMILY_NRF
```

9.3 Building and Running

copy the samples/gui/clock from this repository to the zephyr one.

```
west build -p -b pinetime samples/gui/clock
```

9.4 Todo

- time of day clock
- setting the time

9.5 References

CHAPTER

TEN

SERIAL NOR FLASH

```
west build -p -b pinetime samples/drivers/spi_flash -DCONF=prj.conf
```

10.1 Overview

This sample application should unlock the serial nor flash memory. This can be very usefull to store e.g. background for the watch.

compilation problematic ....

/root/zephyrproject/zephyr/samples/drivers/spi_flash/src/main.c:17:22: error: 'DT_INST_0_JEDEC_SPI_NOR_LABEL' undeclared (first use in this function); did you mean 'DT_INST_0_NORDIC_NRF_RTC_LABEL'?

Turns out this is some problem with the board definition file.

I found it to be very useful to consult the generated dts file. Here you can check if everything is present.

Guess the dts-file has to be well intended.(structured)

```
vi /root/zephyrproject/zephyr/build/zephyr/include/generated/generated_dts_board.conf
```

10.2 Requirements

complement the pinetime.dts file with the following (under spi) #define JEDEC_ID_MACRONIX_MX25L64 0xC22017

```
&spi0 {
  compatible = "nordic,nrf-spi";
  status = "okay";
  sck-pin = <2>;
  mosi-pin = <3>;
  miso-pin = <4>;
  cs-gpios = <&gpio0 27 0>,<&gpio0 5 0>;
  st7789v@0 {
          compatible = "sitronix,st7789v";
          label = "DISPLAY";
          spi-max-frequency = <8000000>;
          reg = <0>;
          cmd-data-gpios = <&gpio0 18 0>;
          reset-gpios = <&gpio0 26 0>;
          width = <240>;
```

(continues on next page)

(continued from previous page)

```
          height = <240>;
          x-offset = <0>;
          y-offset = <0>;
          vcom = <0x19>;
          gctrl = <0x35>;
          vrhs = <0x12>;
          vdvs = <0x20>;
          mdac = <0x00>;
          gamma = <0x01>;
          colmod = <0x05>;
          lcm = <0x2c>;
          porch-param = [0c 0c 00 33 33];
          cmd2en-param = [5a 69 02 01];
          pwctrl1-param = [a4 a1];
          pvgam-param = [D0 04 0D 11 13 2B 3F 54 4C 18 0D 0B 1F 23];
          nvgam-param = [D0 04 0C 11 13 2C 3F 44 51 2F 1F 1F 20 23];
          ram-param = [00 F0];
          rgb-param = [CD 08 14];

  };

  mx25r64: mx25r6435f@1 {
          compatible = "jedec,spi-nor";
          reg = <1>;
          spi-max-frequency = <1000000>;
          label = "MX25R64";
          jedec-id = [0b 40 16];
          size = <67108864>;
          has-be32k;
  };
```

10.3 Building and Running

```
west build -p -b pinetime samples/drivers/spi_flash
```

10.4 Todo

- detect ID memory : it is not the macronix one as suggestion on the pinetime website

I found the following : jedec-id = [0b 40 16]; (OK: can execute sample program)

- create working board definition (OK: see above)

10.5 References

http://files.pine64.org/doc/datasheet/pinetime/MX25L6433F,%203V,%2064Mb,%20v1.6.pdf

CHAPTER

ELEVEN

SENSORS ON THE I2C BUS

0x18: Accelerometer: BMA423-DS000 https://github.com/BoschSensortec/BMA423-Sensor-API

0x44: Heart Rate Sensor: HRS3300_Heart

0x15: Touch Controller: Hynitron CST816S Touch Controller

CHAPTER

TWELVE

CONFIGURING I2C

12.1 board level definitions

```
under boards/arm/pinetime are the board definitions
- pinetime.dts
- pinetime_defconfig


The sensors in the pintime use the I2C bus.

&i2c1 {
        compatible = "nordic,nrf-twi";
        status = "okay";
        sda-pin = <6>;
        scl-pin = <7>;

      };
```

12.2 development trajectory

The final goal is to use the accel-sensor in the watch (BMA423), which does not exist yet. In order to minimize the effort:

- we'll use something that looks like it (ADXL372), because there exists an example.
- next we adapt it to use the existing BMA280 sensor (under drivers/sensor)
- finally we create a driver for the BMA423, based upon the BMA280

12.3 defining an I2C sensor

```
under samples/sensor/axl372 we create : "pinetime.overlay"
&i2c1 {
    status = "okay";
    clock-frequency = <I2C_BITRATE_STANDARD>;
    adxl372@18 {
                compatible = "adi,adxl372";
                reg = <0x18>;
                label = "ADXL372";
```

(continues on next page)

(continued from previous page)

```
                int1-gpios = <&gpio0 8 0>;
                };
    };
```

note: this gets somehow merged with the board definition pinetime.dts

```
In the "prj.conf" file we define the sensor

CONFIG_STDOUT_CONSOLE=y
CONFIG_LOG=y
CONFIG_I2C=y
CONFIG_SENSOR=y
CONFIG_ADXL372=y
CONFIG_ADXL372_I2C=y
CONFIG_SENSOR_LOG_LEVEL_WRN=y
```

note: this gets somehow merged with the board definition pinetime_defconfig

12.4 compiling the sample

```
west build -p -b pinetime samples/sensor/adxl372 -DCONF=prj.conf
```

CHAPTER

THIRTEEN

BOSCH BMA280

```
west build -p -b pinetime samples/drivers/bma280
```

13.1 Overview

This sample application mimics the presence of a bosch, bma280 accel sensor. For this sensor exists a driver in zephyr, but no sample. Remember, I'm not a zephyr expert and am learning on the way.

13.2 Requirements

complement the pinetime.dts file with the following (under samples/sensor/bma280)

```
&i2c1 {
        bma280@18 {
                    compatible = "bosch,bma280";
                    reg = <0x18>;
                    label = "BMA280";
                    int1-gpios = <&gpio0 8 0>;
                   };
       };
```

Create a file: */dts/bindings/sensor/bosch,bma280-i2c.yaml*. Which contains:

```
compatible: "bosch,bma280"
include: i2c-device.yaml
properties:
       int1-gpios:
       type: phandle-array
       required: false
```

13.3 Building and Running

13.4 Todo

- since no serial port and no J-LINK, I have to print messages to the screen or trough bluetooth serial (which does not exist, or I haven't found it yet ;))

13.5 References

CHAPTER

FOURTEEN

TOUCHSCREEN HYNITRON

```
git clone https://github.com/lupyuen/hynitron_i2c_cst0xxse
```

14.1 Overview

this does not exist yet in zephyr, but there is work in progress https://github.com/zephyrproject-rtos/zephyr/pull/16119

14.2 Requirements

complement the pinetime.dts file with the following (under samples/sensor/bma280)

```
&i2c1 {
        touch@18 {
                  };
       };
```

Create a file: */dts/bindings/sensor/touch.yaml*. Which contains:

```
compatible: "touch"
include: i2c-device.yaml
properties:
       int1-gpios:
       type: phandle-array
       required: false
```

14.3 Building and Running

14.4 Todo

-create touchscreen driver -create sample

14.5 References

CHAPTER

FIFTEEN

MENUCONFIG

15.1 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 configurationfile (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)

```
-*- Entropy Drivers  --->
[*] GPIO Drivers  --->
[ ] Shared interrupt driver  ----
[ ] SPI hardware bus support  ----
[ ] I2C Drivers  ----
[ ] I2S bus drivers  ----
[ ] PWM (Pulse Width Modulation) Drivers  ----
[ ] Enable board pinmux driver  ----
[ ] ADC drivers  ----
[ ] Watchdog Support  ----
[*] Hardware clock controller support  ---> <<<<<<<<<<<<<<<<<SELECT THIS ONE<<<<<<<<<<
[ ] Precision Time Protocol Clock driver support
[ ] IPM drivers  ----
    Max compiled-in log level for ipm (Info)  --->
[ ] Flash hardware support  ----
[ ] Sensor Drivers  ----
```

```
    Max compiled-in log level for clock control (Info)  --->
[*] NRF Clock controller support  ---> <<<<<<<<<<<<<<<<<<<<SELECT THIS ONE<<<<<<<<<<
```

CHAPTER

SIXTEEN

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 be able 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. (my repo: https://github.com/najnesnaj/openocd is adapted for the orange pi)

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*

CHAPTER

SEVENTEEN

DEBUGGING THE PINETIME SMARTWATCH

```
The pinetime does not have a serial port.
I do not have a segger debugging probe.
A way around this, it to put a value in memory at a fixed location.
With openocd you can peek at this memory location.
```

Note: #define MY_REGISTER (*(volatile uint8_t*)0x2000F000)

in the program you can set values: MY_REGISTER=1; MY_REGISTER=8;

this way you know till where the code executes

```
#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
>mdw 0x2000F000 0x1

the last byte shows the value of your program trace value
```

CHAPTER

EIGHTEEN

SCANNING THE I2C_1 PORT

```
The pinetime does not have a serial port.
I do not have a segger debugging probe.
A way around this, it to put a value in memory at a fixed location.
With openocd you can peek at this memory location.
```

18.1 Building and Running

In this repo under samples you will find an adapted i2c scanner program.

```
west build -p -b pinetime samples/drivers/i2c_scanner
```

Note: #define MY_REGISTER (*(volatile uint8_t*)0x2000F000)

in the program you can set values: MY_REGISTER=1; MY_REGISTER=8;

this way you know till where the code executes

```
#telnet 127.0.0.1 4444
```

Peeking

```
once your telnet sessions started:
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Open On-Chip Debugger
>mdw 0x2000F000 0x1
0x2000f000: 00c24418
```

Note::

this corresponds to 0x18, 0x44 and 0xC2 (which is endvalue of scanner, so it does not detect touchscreen, which should be touched first. . . .)

CHAPTER

NINETEEN

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

CHAPTER

TWENTY

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
```

CHAPTER

TWENTYONE

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 then me.
You are invited to add the missing pieces and to improve what's already there.
```

21.1 Todo

list with suggestions:

- better graphics (lvgl using images and rotating stuff)
- NOR flash (here one can store data)
- 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 . . .

21.2 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)