open source watch Documentation

Release 1.0.0

jj

Feb 24, 2020

CONTENTS

Note : You may at any time read the book, store it in your ereaders

CHAPTER

ONE

AUTHOR:

Jan Jansen najnesnaj@yahoo.com

CHAPTER

TWO

LICENSE:

All the software is subject to the Apache 2.0 license (same as zephyr), which is very liberal.

CHAPTER

THREE

ZEPHYR FOR THE PINETIME SMARTWATCH

```
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 (pinetime works as an external (out of tree)
      application for zephyr)
    - try out the samples
    - inspire you to modify and enhance
```

suggestion :

- follow the installation instructions
- try some examples
- try out bluetooth
- try out the display

CHAPTER

FOUR

INSTALL ZEPHYR

4.1 In case you already have zephyr installed:

Pinetime works as external (out of tree) application. You can clone pinetime next to zephyr in the working directory and update manifest and west.

```
west config manifest.path pinetime
```

4.2 In case you start from scratch :

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

In the Zephyr getting started page :

1) select and update OS
2) install dependencies
3) Get the source code

```
instead of following the procedure:
       cd ~
       west init zephyrproject
       cd zephyrproject
       west update

you should do this :
        cd ~
        mkdir work
        cd work
        west init -m https://github.com/najnesnaj/pinetime-zephyr
        west update
```

4) complete the other steps

to test if your install works :

cd ~/work/pinetime

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

TIP : sometimes you run into trouble compiling: removing the build directory can help in that case

CHAPTER

FIVE

STARTING WITH SOME BASIC APPLICATIONS

The best way to get a feel of zephyr for the PineTime watch, is to start building applications.

The gpio ports, i2c communication, memory layout, stuff that is particular for the watch is defined in the board definition file.

The provided samples are standard zephyr application, with some minor modifications.

5.1 Blinky example

```
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-LCD remains black.

The screen contains three leds, this way the intensity is set.
```

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

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

see : *Blinky Application*

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

once the compilation is completed, you can find the firmware under : ~/work/pinetime/build/zephyr/zephyr.bin

5.2 Reading out the button on the watch

```
The pinetime does have a button on the side.

In order to check if the button is pressed, it sets a value in memory.
With openocd you can peek at this memory location.
```

5.2.1 Building and Running

see : *Button demo*

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

```
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)
```

CHAPTER

SIX

BLUETOOTH (BLE) EXAMPLE

The PineTime uses a Nordic nrf52832 chip, which has BLE functionality build into it.

To test, you can compile a standard application : Eddy Stone.

The watch will behave as a bluetooth beacon, and you should be able to detect it with your smartphone or with bluez under linux.

6.1 Using a standard zephyr application under pinetime:

Each sample has its own directory. In this directory you will notice a file : "CMakeLists.txt".

In order to use a standard, you can just copy it under the pinetime directory.

In order to be able to compile it, you just have to add one line in the CMakeList.txt :

include($ENV{ZEPHYR_BASE}/../pinetime/cmake/boilerplate.cmake)

Have a look in the samples/bluetooth/eddystone directory.

6.2 Eddy Stone

see: *Bluetooth: Eddystone*

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

6.3 Using the created bluetooth sample:

I use linux with a bluetoothadapter 4.0. You need to install 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.

6.4 Ble Peripheral

this example is a demo of the services under bluetooth

first build the image

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

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

6.5 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

SEVEN

DISPLAY (ST7789)

7.1 Display example

This is just a simple display test. It displays coloured squares, but it allows you to check if the screen is OK.

TIP: While connecting 5V, do not connect 3.3V at the same time

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

They need to be on (LOW) to visualize the display.
Have a look in the source code.
```

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

Once the compilation is completed you can upload the firmware.

If all goes well, you should see some coloured squares on your screen.

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

EIGHT

GFX LIBRARY SAMPLE

8.1 Overview

This sample is built on top of the ST7789 display sample (*display (st7789)*), extending it with the Adafruit GFX Library. The library was ported from Arduino and has the same functionality and API. See `src/main.cpp` for examples on the GFX API usage.

See *display (st7789)* for more details on working with the display itself.

8.2 Usage

Add the gfx sample from this repo into your project:

```
$ cp samples/gui/gfx ~/zephyrproject/zephyr/samples/gui/
```

Note: In order to make the library work the sample is built with C++ support. This is achieved by having the following line in the sample's *prj.conf* configuration:

```
CONFIG_CPLUSPLUS=y
```

Build & flash the sample:

```
$  west build -p -b pinetime samples/gui/gfx
$  west flash
```

If all goes well, you should see a looping graphical test: drawing lines, rectangles, triangles etc.

CHAPTER

NINE

LITTLEVGL BASIC SAMPLE

9.1 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.

9.2 Requirements

The program has been modified to light up the background leds.

TIP: matching label : DISPLAY

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

9.3 Building and Running

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

9.3.1 modifying the font size :

```
west build -t menuconfig
```

goto:

- additional libraries
- lvgl gui library

(look for fonts, and adapt according to your need)

9.3.2 apply changes of the changed config:

```
west build
```

(instead of west build -p (pristine) which wipes out your customisation)

9.4 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...

9.5 References

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

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

CHAPTER

TEN

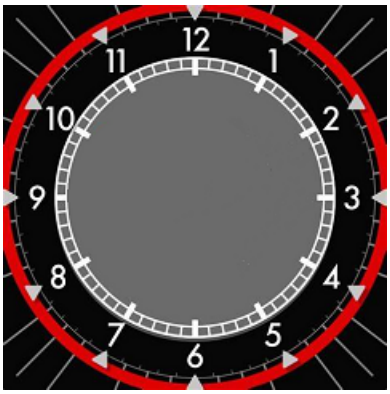
LITTLEVGL CLOCK SAMPLE

see : *LittlevGL Clock Sample*

10.1 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.



10.2 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

10.3 Building and Running

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

10.4 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...

10.5 References

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

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

CHAPTER

ELEVEN

REAL TIME CLOCK

11.1 Overview

This sample application “clock” uses the RTC0 timer. It uses the counter driver. (based on the alarm sample)

Basically an interrupt is set to go off after 1 second. The number of seconds is incremented and the interrupt is launched again.

It will serve as a building block for a “time of the day” clock.

In addition it will need a function to set the time.

In bluetooth one can use CTS (central time service)

**NOTE: as I found out there is a conflict between RTC0 and bluetooth **

An example of using RTC2 is included

11.2 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
```

see : *LittlevGL Clock Sample*

11.3 Building and Running

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

11.4 Todo

- time of day clock

- setting the time

11.5 References

CHAPTER

TWELVE

CURRENT TIME SERVICE

https://www.bluetooth.com/specifications/gatt/services/ https://www.bluetooth.com/specifications/gatt/characteristics/ 0x1805 current time service 0x2A2B current time characteristic

12.1 Requirements:

You need :

- **a CTS server (use of bluez on linux explained)**
  - start the CTS service (python script)
  - connect to the CTS client
- a CTS client (the pinetime watch)

12.2 BLE Peripheral CTS sample for zephyr

This example demonstrates the basic usage of the current time service. It is based on the https://github.com/Dejvino/pinetime-hermes-firmware. It starts advertising it's UUID, and you can connect to it. Once connected, it will read the time from your CTS server (bluez on linux running the gatt-cts-server script in my case)

first build the image

```
$  west build -p -b pinetime samples/bluetooth/peripheral-cts
```

12.3 Using bluez on linux to connect

The pinetime zephyr sample behaves as a peripheral:

- first of all start the cts service

  -connect to the pinetime with bluetoothctl

Using bluetoothctl:

```
#bluetoothctl
[bluetooth]#scan on


[NEW] Device 60:7C:9E:92:50:C1 Zephyr Peripheral Sample Long
```

(continues on next page)

(continued from previous page)

```
once you see your device
[blueooth]#connect 60:7C:9E:92:50:C1 (the device mac address as displayed)
```

12.4 Howto use Bluez on linux to set up a time service

Within the bluez source distribution there is an example GATT (Generic Attribute Profile)server. It advertises some standard service such as heart rate, battery ... Koen zandberg adapted this script, so it advertises the current time : https://github.com/bosmoment/gatt-cts/blob/master/gatt-cts-server.py

You might have to install extra packages:

```
apt-get install python-dbus
apt-get install python-gi
apt-get install python-gobject
```

12.5 Howto use Android to set up a time service

As soon as a device is bonded, Pinetime will look for a CTS server (Current Time Service) on the connected device. Here is how to do it with an Android smartphone running NRFConnect:

Build and program the firmware on the Pinetime Install NRFConnect (https://www.nordicsemi.com/Software-and-Tools/Development-Tools/nRF-Connect-for-desktop)

Start NRFConnect and create a CTS server : Tap the hamburger button on the top left and select "Configure GATT server" Tap "Add service" on the bottom Select server configuration "Current Time Service" and tap OK Go back to the main screen and scan for BLE devices. A device called "PineTime" should appear Tap the button "Connect" next to the PineTime device. It should connect to the PineTime and switch to a new tab. On this tab, on the top right, there is a 3 dots button. Tap on it and select Bond. The bonding process begins, and if it is sucessful, the PineTime should update its time and display it on the screen.

CHAPTER

THIRTEEN

DRIVERS

13.1 configuring I2C

13.1.1 board level definitions

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


The sensors in the pinetime use the I2C bus.

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

      };
```

13.1.2 definition on project level

In the directory of a sample, you will find a prj.conf file. Here you can set values specific for you project/sample.

```
In the "prj.conf" file we define the sensor (eg adxl372)

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 (overlayed) with the board definition pinetime_defconfig

13.2 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

13.3 Bosch BMA421

this driver does not exist, so it has been created. Still work in progress ....

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

13.3.1 Overview

BMA421 is not a part number available to the general public, and therefore all the supporting documentation and design resources are neither discussed in public forums, nor disclosed on GitHub.

CHIP_ID=0X11 (so the Bosch BMA423 drivers need to be adapted)

The Bosch documentation on the bma423 seems to apply to the bma421.

13.3.2 Requirements

for this sensor does not exist any driver, so here's what I did to create one under zephyr

adapt CMakeLists.txt

~/zephyrproject-2/zephyr/drivers/sensor add_subdirectory_ifdef(CONFIG_BMA280 bma280) add_subdirectory_ifdef(CONFIG_BMA421 bma421)

adapt Kconfig

~/zephyrproject-2/zephyr/drivers/sensor

add yaml file

~/zephyrproject-2/zephyr/dts/bindings/sensor cp bosch,bma280-i2c.yaml bosch,bma421-i2c.yaml

edit KConfig

source "drivers/sensor/bma280/Kconfig" source "drivers/sensor/bma421/Kconfig"

source "drivers/sensor/bmc150_magn/Kconfig"

source "drivers/sensor/bme280/Kconfig"

create driver

see under drivers/sensor/bma421

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

```
&i2c1 {
        bma421@18 {
                    compatible = "bosch,bma421";
                    reg = <0x18>;
                    label = "BMA421";
                    int1-gpios = <&gpio0 8 0>;
                   };
       };
```

Create a file: */dts/bindings/sensor/bosch,bma421-i2c.yaml*. Which contains:

```
compatible: "bosch,bma421"
include: i2c-device.yaml
properties:
       int1-gpios:
       type: phandle-array
       required: false
```

13.3.3 Building and Running

13.3.4 Todo

- the driver is interrupt driven as well – need to test software
- the sensor has algorithm for steps – read out register
- temperature some attempt has been made, but . . . (OK, temp can be read)

13.3.5 References

Bosch has documented the BMA423 very well. I kind of hope it will apply to the bma421.

A mechanism to adapt the 0x5E register is provided. (burst read/write)

All kind of parameters can be set to trigger an interrupt. (e.g. number of steps taken : think of the 10000 steps threshold)

13.4 HYNITRON CST816S

this driver does not exist, so it has been created. Still work in progress . . . .

there is a sample in this repository which can be copied to the zephyr samples directory

```
west build -p -b pinetime samples/sensor/cst816s
```

13.4.1 Overview

the Hynitron cst816s is a touchscreen. In zephyr doesn't handle touchscreens yet. In order to investigate, the touchscreen driver has been created as a sensor. In fact it senses your finger ;)

13.4.2 Requirements

for this sensor does not exist any driver, so here's what I did to create one under zephyr

adapt CMakeLists.txt adapt Kconfig add yaml file

create driver

The driver reads only one position. Multitouch is possible, but the screen is small. . . .

see under drivers/sensor/cst816s

have a look at the pinetime.dts (under board/arm/pinetime) file:

```
&i2c1 {
        cst816s@15 {
                   compatible = "hynitron,cst816s";
                   reg = <0x15>;
                   label = "CST816S";
                   };
       };
```

13.4.3 Building and Running

There are two samples :

- samples/gui/lvtouch (graphical)
- samples/sensor/cst816s (no graphics)

13.4.4 Todo

The graphical sample doesn't handle interrupts.

13.4.5 References

There is little available for this touchscreen.

13.5 HX HRS3300

this driver does not exist, so it has been created. Still work in progress . . . .

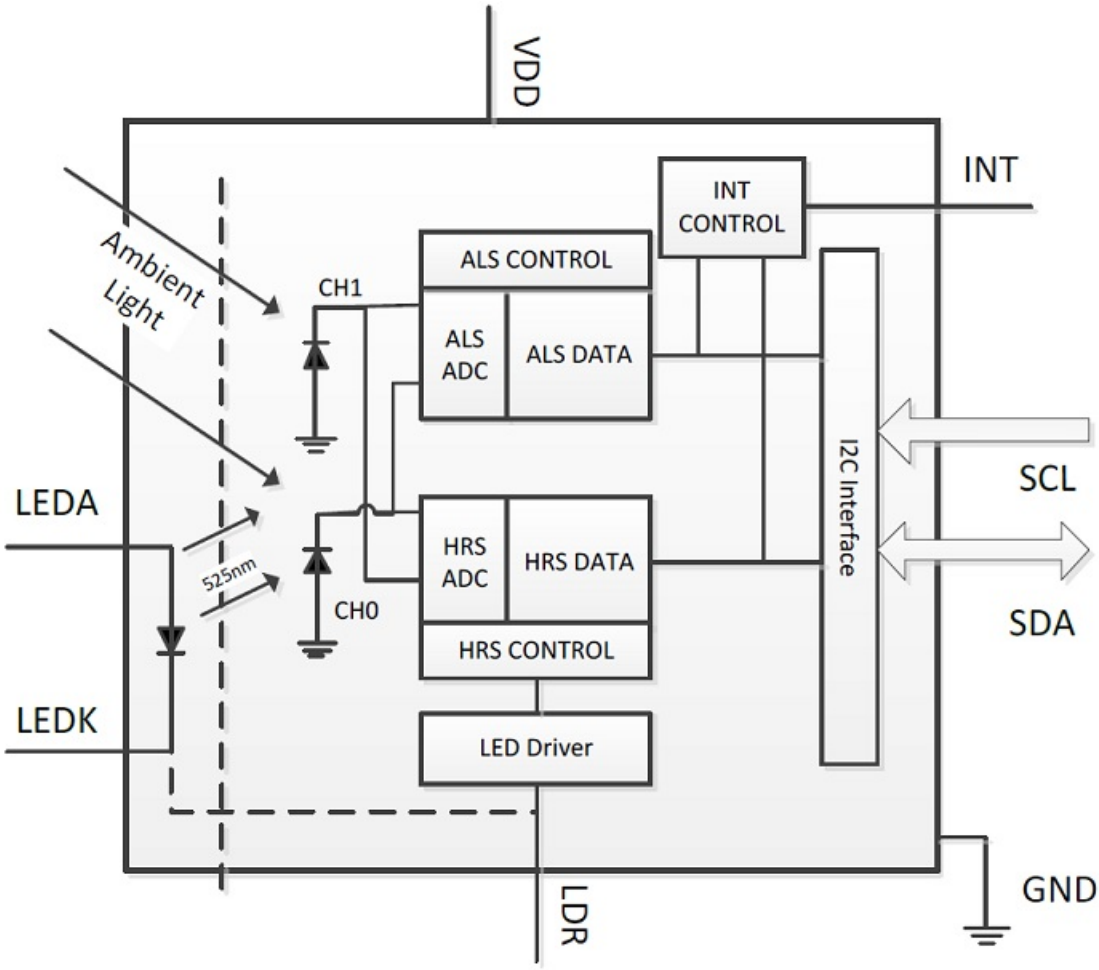
there is a sample in this repository which can be copied to the zephyr samples directory

```
west build -p -b pinetime samples/sensor/hrs3300
```

13.5.1 Overview

The HX HRS3300 sensor is a heart rate sensor, it produces 2 values: ALS and HRS. Ambient LIGHT SENSOR and HEART RATE SENSOR. Which have to be processed by an algorithm. I have no knowledge of a good open source algorithm yet.

I have used the settings of an arduino port of this library.



13.5.2 Requirements

for this sensor does not exist any driver, so here's what I did to create one under zephyr

adapt CMakeLists.txt

~/zephyrproject-2/zephyr/drivers/sensor add_subdirectory_ifdef(CONFIG_HRS3300 hrs3300)

adapt Kconfig

~/zephyrproject-2/zephyr/drivers/sensor

add yaml file

~/zephyrproject-2/zephyr/dts/bindings/sensor add hx,hrs3300.yaml

edit KConfig

source “drivers/sensor/hrs3300/Kconfig”

create driver

see under drivers/sensor/hrs3300

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

```
&i2c1 {
        hrs3300@44 {
                   compatible = "hx,hrs3300";
                   reg = <0x44>;
                   label = "HRS3300";
                   };
       };
```

Create a file: */dts/bindings/sensor/hx,hrs3300.yaml*. Which contains:

```
compatible: "hx,hrs3300"
properties:
```

13.5.3 Building and Running

13.5.4 Todo

- algorithm for heartrate
- power saving
- switching off/on mechanism

13.5.5 References

HRS3300 Heart Rate Sensor.pdf https://github.com/atc1441/HRS3300-Arduino-Library

13.6 Serial Nor Flash

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

13.6.1 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)

**TIP: consult the generated dts board file **

consulting the generated board definition file

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

13.6.2 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>;
          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];
```

(continues on next page)

(continued from previous page)

```
        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;
};
```

13.6.3 Building and Running

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

13.6.4 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)

13.6.5 References

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

13.7 Battery

the samples just gets an analog reading from the battery

```
west build -p -b pinetime samples/sensor/battery
```

13.7.1 Overview

The battery level is measured on port 31, trough an ADC conversion.

voltage = (value * 6)/1024 percentage remaining ((voltage - 3.55)*100)*3.9;

A module should be able to report battery status in milivolts and charge level in percentage. Additionally, it should notify when external power is connected and when battery is being charged. Module will use adc (saadc peripheral) to measure battery voltage and gpio driver to monitor charge indication pin (pin 0.12) and power presence pin (0.19). Battery voltage can be in range from 3.0V - 4.2V (?). Unfortunately, internal reference (0.6V) can only be used for

voltages up to 3.6V (due to minimal gain of 1/6). VDD/4 reference can be used with 1/6 gain to measure voltages up to 4.95V. Test is needed to check how accurate is VDD as reference. Discharge curve (https://forum.pine64.org/showthread.php?tid=8147) will be used to calculate charge level in percent. Things to consider: saadc periodical calibration (spec suggests calibration if temperature changes by 10'C) inaccuracy of results: oversampling? never report higher level than before (if charge not connected), etc.

13.7.2 Todo

check pin when charging

13.7.3 References

https://forum.pine64.org/showthread.php?tid=8147

CHAPTER

FOURTEEN

FIRMWARE OVER THE AIR (FOTA)

14.1 Wireless Device Firmware Upgrade

14.1.1 Overview

In order to perform a FOTA (firmware over the air) update on zephyr you need 2 basic components:

- MCUboot (a bootloader)
- SMP Server (a bluetooth service)

14.2 MCUboot with zephyr

Clone MCUBOOT for zephyr from github. Install additional packages required for development with mcuboot:

```
cd ~/mcuboot  # or to your directory where mcuboot is cloned
pip3 install --user -r scripts/requirements.txt
```

To build MCUboot, create a build directory in boot/zephyr, and build it as follows:

```
cd boot/zephyr
mkdir build && cd build
cmake -GNinja -DBOARD=pinetime ..
ninja
```

After building the bootloader, the binaries should reside in *build/zephyr/zephyr.{bin,hex,elf}*.

This image can be flashed as a normal application.

Some additional configuration is required to build applications for MCUboot.

This is handled internally by the Zephyr configuration system and is wrapped in the *CONFIG_BOOTLOADER_MCUBOOT* Kconfig variable, which must be enabled in the application's *prj.conf* file.

The Zephyr *CONFIG_BOOTLOADER_MCUBOOT* configuration option [documentation](http://docs.zephyrproject.org/reference/kconfig/CONFIG_BOOTLOADER_MCUBOOT.html) provides additional details regarding the changes it makes to the image placement and generation in order for an application to be bootable by MCUboot.

In order to upgrade to an image (or even boot it, if *MCUBOOT_VALIDATE_PRIMARY_SLOT* is enabled), the images must be signed.

To make development easier, MCUboot is distributed with some example keys. It is important to stress that these should never be used for production, since the private key is publicly available in this repository. See below on how to make your own signatures.

Images can be signed with the *scripts/imgtool.py* script. It is best to look at *samples/zephyr/Makefile* for examples on how to use this.

Since the bootloader is already in place, you cannot flash your application.bin to 0x00000.

Eg. in openocd : program application.bin 0x0c000. (which corresponds to the flash layout of slot 0)

These images can also be marked for upgrade, and loaded into the secondary slot, at which point the bootloader should perform an upgrade.

14.3 Partitions

```
have a look at boards/arm/pinetime/pinetime.dts
```

14.3.1 Defining partitions for MCUboot

The first step required for Zephyr is making sure your board has flash partitions defined in its device tree. These partitions are:

- *boot_partition*: for MCUboot itself
- *image_0_primary_partition*: the primary slot of Image 0
- *image_0_secondary_partition*: the secondary slot of Image 0
- *scratch_partition*: the scratch slot

The flash partitions are defined in the pinetime boards folder, in a file named *boards/arm/pinetime/pinetime.dts*.

14.3.2 Using NOR flash in partitions

The flash space on the Nordic nrf52 is 512K. Basically with the partitioning you end up with less space for your program.

As the pinetime has an extra spi nor flash chip, we can use this.

The flashlay-out can be modified so as 1 chunk is on system flash and 1 chunk is on SPI NOR flash. This way the space for your firmware remains almost the same.

14.4 Signing an application

In order to improve the security, only signed images can be uploaded.

There is a public and private key. The Bootloader is compiled with the public key. Each time you want to upload firmware, you have to sign it with a private key.

NOTE: it is important to keep the private key hidden

14.4.1 Generating a new keypair

Generating a keypair with imgtool is a matter of running the keygen subcommand:

```
$ ./scripts/imgtool.py keygen -k mykey.pem -t rsa-2048
```

14.4.2 Extracting the public key

The generated keypair above contains both the public and the private key. It is necessary to extract the public key and insert it into the bootloader.

```
$ ./scripts/imgtool.py getpub -k mykey.pem
```

This will output the public key as a C array that can be dropped directly into the *keys.c* file.

14.4.3 Example

sign the compiled zephyr.bin firmware with the root-rsa-2048.pem, private key:

```
imgtool.py sign --key ../../root-rsa-2048.pem \
    --header-size 0x200 \
    --align 8 \
    --version 1.2 \
    --slot-size 0x60000 \
    ../mcuboot/samples/zephyr/build/ds_d6/hello1/zephyr/zephyr.bin \
    signed-hello1.bin
```

14.5 SMP Server Sample

14.5.1 Overview

This sample application implements a Simple Management Protocol (SMP) server. SMP is a basic transfer encoding for use with the MCUmgr management protocol. For more information about MCUmgr and SMP, please see device_mgmt.

This sample application supports the following mcumgr transports by default:

- Shell
- Bluetooth

14.5.2 Requirements

In order to communicate with the smp server sample installed on your pinetime, you need mcumgr.

Here is a procedure to install mcumgr on a raspberry pi (or similar)

14.5.3 Building and Running

Step 1: Build smp_svr

`smp_svr` can be built for the nRF52 as follows:

Step 2: Sign the image

Using MCUboot's `imgtool.py` script, sign the `zephyr.(bin|hex)` file you built in Step 3. In the below example, the MCUboot repo is located at `~/src/mcuboot`.

```
~/src/mcuboot/scripts/imgtool.py sign \
     --key ~/src/mcuboot/root-rsa-2048.pem \
     --header-size 0x200 \
     --align 8 \
     --version 1.0 \
     --slot-size <image-slot-size> \
     <path-to-zephyr.(bin|hex)> signed.(bin|hex)
```

The above command creates an image file called `signed.(bin|hex)` in the current directory.

Step 3: Flash the smp_svr image

Upload the bin-file from Step 2 to image slot-0. For the pinetime, slot-0 is located at address `0xc000`.

```
in openocd : program zephyr.bin 0xc000
```

Step 4: Run it!

Note: If you haven't installed `mcumgr` yet, then do so by following the instructions in the mcumgr_cli section of the Management subsystem documentation.

The `smp_svr` app is ready to run. Just reset your board and test the app with the `mcumgr` command-line tool's `echo` functionality, which will send a string to the remote target device and have it echo it back:

```
sudo mcumgr --conntype ble --connstring ctlr_name=hci0,peer_name='Zephyr' echo hello
hello
```

Step 5: Device Firmware Upgrade

Now that the SMP server is running on your board and you are able to communicate with it using `mcumgr`, you might want to test "OTA DFU", or Over-The-Air Device Firmware Upgrade.

To do this, build a second sample (following the steps below) to verify it is sent over the air and properly flashed into slot-1, and then swapped into slot-0 by MCUboot.

Build a second sample

Perhaps the easiest sample to test with is the **:zephyr_file:'samples/hello_world'** sample provided by Zephyr, documented in the hello_world section.

Edit **:zephyr_file:'samples/hello_world/prj.conf'** and enable the required MCUboot Kconfig option as described in mcuboot by adding the following line to it:

```
CONFIG_BOOTLOADER_MCUBOOT=y
```

Then build the sample as usual (see hello_world).

Sign the second sample

Next you will need to sign the sample just like you did for `smp_svr`, since it needs to be loaded by MCUboot. Follow the same instructions described in smp_svr_sample_sign, but this time you must use a `.bin` image, since `mcumgr` does not yet support `.hex` files.

Upload the image over BLE

Now we are ready to send or upload the image over BLE to the target remote device.

```
sudo mcumgr --conntype ble --connstring ctlr_name=hci0,peer_name='Zephyr' image␣
↪upload signed.bin
```

If all goes well the image will now be stored in slot-1, ready to be swapped into slot-0 and executed.

Note: At the beginning of the upload process, the target might start erasing the image slot, taking several dozen seconds for some targets. This might cause an NMP timeout in the management protocol tool. Use the `-t <timeout-in-seconds` option to increase the response timeout for the `mcumgr` command line tool if this occurs.

List the images

We can now obtain a list of images (slot-0 and slot-1) present in the remote target device by issuing the following command:

```
sudo mcumgr --conntype ble --connstring ctlr_name=hci0,peer_name='Zephyr' image list
```

This should print the status and hash values of each of the images present.

Test the image

In order to instruct MCUboot to swap the images we need to test the image first, making sure it boots:

```
sudo mcumgr --conntype ble --connstring ctlr_name=hci0,peer_name='Zephyr' image test
↪<hash of slot-1 image>
```

Now MCUBoot will swap the image on the next reset.

Reset remotely

We can reset the device remotely to observe (use the console output) how MCUboot swaps the images:

```
sudo mcumgr --conntype ble --connstring ctlr_name=hci0,peer_name='Zephyr' reset
```

Upon reset MCUboot will swap slot-0 and slot-1.

The new image is the basic `hello_world` sample that does not contain SMP or BLE functionality, so we cannot communicate with it using `mcumgr`. Instead simply reset the board manually to force MCUboot to revert (i.e. swap back the images) due to the fact that the new image has not been confirmed.

If you had instead built and uploaded a new image based on `smp_svr` (or another BLE and SMP enabled sample), you could confirm the new image and make the swap permanent by using this command:

```
sudo mcumgr --conntype ble --connstring ctlr_name=hci0,peer_name='Zephyr' image␣
↪confirm
```

Note that if you try to send the very same image that is already flashed in slot-0 then the procedure will not complete successfully since the hash values for both slots will be identical.

CHAPTER

FIFTEEN

BEHIND THE SCENE

15.1 Behind the scene

15.1.1 Overview

I'm not a zephyr expert and am learning on the way.

In this chapter I let you glimpse behind the scene. (and notice all the struggle)

In case of the accel sensor, I used the bosch bma280 as a template.

In case of the touchscreen, I soon ran into trouble. So I tried splitting a complex problem into simpler ones.

This allowed me to detect problems easier.

The created samples might be of some use, if you run into trouble or if you want to extend the functionality.

15.2 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), so we get a sample that works with the BMA280.
- we create a driver for the BMA423, based upon the BMA280
- we adapt the sample for the BMA280 to BMA423

15.3 Bosch BMA280

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

15.3.1 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.

15.3.2 Requirements

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

This will supplement / override what's already definition on the board level (boards/arm/pinetime)

```
&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
```

15.3.3 Building and Running

15.3.4 Todo

- since no serial port and no J-LINK, I have to print messages to the screen (see sample gui/lvaccel)
- I adapted the BMA driver so it accepts the CHIP_ID, further registers are subject to investigation, since no doc

15.3.5 References

BMA421 is not a part number available to the general public, and therefore all the supporting documentation and design resources are neither discussed in public forums, nor disclosed on GitHub.

CHIP_ID=0X11 (so 423 drivers need to be adapted)

15.4 Touchscreen Hynitron

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

15.4.1 Overview

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

15.4.2 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
```

15.4.3 Building and Running

15.4.4 Todo

-create touchscreen driver -create sample

15.4.5 References

15.5 Troubleshooting drivers

Drivers, like the one for the accel sensor BMA421 or the touchscreen CST816S, can deal with interrupts.

Adapting existing drivers did not get me the desired quick results.

Even after analysing the behaviour, setting values at each function step, did not get me any further.

15.5.1 Overview

The drivers can use interrupts.

In the settings/config one can choose between OWN_THREAD and GLOBAL_THREAD.

This affect the behaviour of how threads are handled.

The tread-handling and interrupt-handling occurs in the driver itself.

An interrupt is handled immediatly, the processing is offloaded to the threading.

15.5.2 Example

- You touch the touchscreen
- the touchscreen generates an interrupt
- the driver handles the interrupt
- a thread is created by the interrupt

- the threadhandling read the I2C-bus

15.5.3 Requirements

In order to create a working driver, I took it apart :

(split a complex problem into simple problems)

a sample to detect interrupt

samples/basic/testirq

Each time the touchscreen gets touched, it increases a counter.

a sample to scan the I2C-BUS

(*scanning the I2C_1 port*),

a sample to read the I2C-BUS

samples/basic/touched It is based on the Hynitron touchscreen code. Mass reading 63 bytes was not possible.

I did add a write of 1 to register 0x00.

a samples to handle semaphores

samples/basic/testsemaphore

15.6 placing a button on the screen

```
This sample is not really important, but it will teach you that you need to set LVGL_
↪CONFIG values, in order    to be able to use LVGL functions.
```

15.6.1 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

```

```

problem the canvas heigh*width eats up RAM and exceeds once > 40

CHAPTER

SIXTEEN

SAMPLES AND DEMOS

In each sample directory is a Readme file. This is just a collection of them.

16.1 Basic Samples

16.1.1 Blinky Application

Overview

The Blinky example shows how to configure GPIO pins as outputs which can also be used to drive LEDs on the hardware usually delivered as "User LEDs" on many of the supported boards in Zephyr.

Requirements

The demo assumes that an LED is connected to one of GPIO lines. The sample code is configured to work on boards that have defined the led0 alias in their board devicetree description file. Doing so will generate these variables:

- DT_ALIAS_LED0_GPIOS_CONTROLLER
- DT_ALIAS_LED0_GPIOS_PIN

Building and Running

This samples does not output anything to the console. It can be built and flashed to a board as follows:

After flashing the image to the board, the user LED on the board should start to blink.

16.1.2 Button demo

Overview

A simple button demo showcasing the use of GPIO input with interrupts. If the button is pressed, it will set a value at the location 0x2000F000 in memory. With openocd or any other debugger you can peek at this location.

Requirements

The demo assumes that a push button is connected to one of GPIO lines. The sample code is configured to work on boards with user defined buttons and that have defined the SW0_* variables.

To use this sample, you will require a board that defines the user switch in its header file. The `board.h` must define the following variables:

- SW0_GPIO_NAME (or DT_ALIAS_SW0_GPIOS_CONTROLLER)
- DT_ALIAS_SW0_GPIOS_PIN

Alternatively, this could also be done by defining 'sw0' alias in the board devicetree description file.

Building and Running

This sample can be built for multiple boards, in this example we will build it for the pinetime

After startup, the program looks up a predefined GPIO device, and configures the pin in input mode, enabling interrupt generation on falling edge. During each iteration of the main loop, the state of GPIO line is monitored and printed to the serial console. When the input button gets pressed, the interrupt handler will print an information about this event along with its timestamp.

16.1.3 I2C Scanner sample

Overview

This sample sends I2C messages without any data (i.e. stop condition after sending just the address). If there is an ACK for the address, it prints the address as `FOUND`.

Warning: As there is no standard I2C detection command, this sample uses arbitrary SMBus commands (namely SMBus quick write and SMBus receive byte) to probe for devices. This sample program can confuse your I2C bus, cause data loss, and is known to corrupt the Atmel AT24RF08 EEPROM found on many IBM Thinkpad laptops. See also the i2cdetect man page

Building and Running

16.1.4 Touchscreen IRQ

Overview

The touchscreen generates an interrupt when touched.

Requirements

A counter that keeps track of the number of times touched.

This value is stored at a fixed location in memory, because I have a simple test setup.

Building and Running

16.1.5 Touchpoints

Overview

When touched the touchscreen triggers an interrupt, it's address 0x15 becomes visable.

Requirements

Cach the interrupts and act upon it.

Only the first touchpoint is usable.

But a sequence of 64 has to be read.

Building and Running

the purpose is just testing howto read the touchpoints of the touchscreen

16.2 Sensor Samples

16.2.1 BMA280: Three Axis High-g I2C/SPI Accelerometer

Description

This sample application produces slightly different outputs based on the chosen driver configuration mode:

- In **Measuring Mode with trigger support**, the acceleration on all three axis is printed in m/s^2 at the sampling rate (ODR).
- In **Polled Measuring Mode**, the instantaneous acceleration is polled every 2 seconds.
- In **Max Peak Detect Mode**, the device returns only the over-threshold peak acceleration between two consecutive sample fetches or trigger events. (In most high-g applications, a single 3-axis acceleration sample at the peak of an impact event contains sufficient information about the event, and the full acceleration history is not required.) Instead of printing the acceleration on all three axis, the sample application calculates the vector magnitude (root sum squared) and displays the result in g's rather than in m/s^2, together with an bar graph.

References

- BMA280: http://www.analog.com/bma280

Wiring

This sample uses the BMA280 sensor controlled either using the I2C or SPI interface. Connect supply **VDD**, **VS** and **GND**. The supply voltage can be in the 1.6V to 3.5V range.

I2C mode

Connect Interface: **SDA**, **SCL** and optionally connect the **INT1** to a interrupt capable GPIO. It is a requirement that **SCLK** must be connected to **GND** in I2C mode. Depending on the baseboard used, the **SDA** and **SCL** lines require Pull-Up resistors. With the **MISO** pin low, the I2C address for the device is 0x1D, and an alternate I2C address of 0x53 can be chosen by pulling the **MISO** pin high.

I2C Address:

- **0x1D**: if MISO is pulled low
- **0x53**: if MISO is pulled high

Note: When sharing an SDA bus, the BMA280 Silicon Revision < 3 may prevent communication with other devices on that bus.

SPI mode

Connect Interface: **SCLK**, **MISO**, **MOSI** and **/CS** and optionally connect the **INT1** to a interrupt capable GPIO.

Building and Running

This project outputs sensor data to the console. It requires an BMA280 sensor. It should work with any platform featuring a I2C/SPI peripheral interface. It does not work on QEMU.

Sample Output: Max Peak Detect Mode

Sample Output: Measurement Mode

16.2.2 CST816S HYNITRON TOUCHSCREEN

Description

When touched the touchscreen triggers an interrupt. This is handled in the driver.

In the sample a handler is defined. This one gets actived by the driver.

References

Wiring

I2C mode

Building and Running

Sample Output: X & Y coordinates

Sample Output: Measurement Mode

```
```

16.2.3 HRS3300 Heart Rate Sensor

Overview

A sensor application that demonstrates how to poll data from the hrs3300 heart rate sensor.

It is based on the max30101 sample.

Building and Running

Sample Output

16.3 Driver Samples

The following samples demonstrate how to use various drivers supported by Zephyr.

16.3.1 I2C Scanner sample

Overview

This sample sends I2C messages without any data (i.e. stop condition after sending just the address). If there is an ACK for the address, it prints the address as `FOUND`.

> **Warning:** As there is no standard I2C detection command, this sample uses arbitrary SMBus commands (namely SMBus quick write and SMBus receive byte) to probe for devices. This sample program can confuse your I2C bus, cause data loss, and is known to corrupt the Atmel AT24RF08 EEPROM found on many IBM Thinkpad laptops. See also the i2cdetect man page

Building and Running

16.3.2 I2C Scanner sample

Overview

This sample sends I2C messages without any data (i.e. stop condition after sending just the address). If there is an ACK for the address, it prints the address as `FOUND`.

> **Warning:** As there is no standard I2C detection command, this sample uses arbitrary SMBus commands (namely SMBus quick write and SMBus receive byte) to probe for devices. This sample program can confuse your I2C bus, cause data loss, and is known to corrupt the Atmel AT24RF08 EEPROM found on many IBM Thinkpad laptops. See also the i2cdetect man page

Building and Running

16.4 Display Samples

16.4.1 ST7789V Display driver

make sure this patch is applied : https://github.com/zephyrproject-rtos/zephyr/pull/20570/files

Overview

This sample will draw some basic rectangles onto the display. The rectangle colors and positions are chosen so that you can check the orientation of the LCD and correct RGB bit order. The rectangles are drawn in clockwise order, from top left corner: Red, Green, Blue, grey. The shade of grey changes from black through to white. (if the grey looks too green or red at any point then the LCD may be endian swapped).

Note: The display driver rotates the display so that the 'natural' LCD orientation is effectively 270 degrees clockwise of the default display controller orientation.

Building and Running

References

- ST7789V datasheet

16.5 GUI Samples

16.5.1 Clock Sample Current Time Service

Overview

This program gets the time of the bluetooth cts server and displays it. Once it gets the time, it uses the RTC2 to generate the time.

Requirements

A cts server. On linux, bluez together with a python script was used.

You need to connect to the pinetime first! (bluetoothctl, connect <MAC>)

Building and Running

References

16.5.2 Clock Sample Current Time Service

Overview

This program gets the time of the bluetooth cts server and displays it.

Requirements

A cts server. On linux, bluez together with a python script was used.

You need to connect to the pinetime first! (bluetoothctl, connect <MAC>)

Building and Running

References

16.5.3 LittlevGL Clock Sample

Overview

This sample application displays a clock background.

This samples demonstrates the use of the counter.

Have a look at the test_counter_interrupt_fn function in src/main.c

Requirements

You have to convert a graphical file to a "C" file, which is like a giant array.

Have a look at the prj.conf file.

It should contain CONFIG_LVGL=y and CONFIG_LVGL_OBJ_IMAGE=y.

For the clock function it needs CONFIG_COUNTER=y.

Building and Running

References

16.5.4 Adafruit GFX Library on ST7789V Display

Overview

This is a sample C++ firmware running Adafruit GFX Library on a ST7789V display. The library is ported from Arduino.

16.5.5 Display accel values

Overview

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

It display the values x,y,z from the bosch BMA421 accel sensor.

Requirements

Pinetime watch definitions can be found under the boards sub-directory

- pinetime.conf
- pinetime.overlay

Building and Running

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

References

16.5.6 LittlevGL Basic Sample

Overview

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

Requirements

Pinetime watch definitions can be found under the boards sub-directory

- pinetime.conf
- pinetime.overlay

Building and Running

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

References

16.5.7 Touchscreen Basic Sample

Overview

This sample application displays touchscreen-values x and y in the center of the screen.

The touchscreen triggers an interrupt when touched. This means that data is ready and can be collected.

However, using the interrupt with the handler, does not seem to work within the GUI. When used in samples/sensor/cst816s, it runs. . . .

Requirements

Pinetime watch zephyr cst816s driver

Building and Running

west build -p -b pinetime samples/gui/lvtouch

References

16.6 Bluetooth Samples

16.6.1 Bluetooth: Central / Heart-rate Monitor

Overview

Similar to the Central sample, except that this application specifically looks for heart-rate monitors and reports the heart-rate readings once connected.

Requirements

- BlueZ running on the host, or
- A board with BLE support

Building and Running

This sample can be found under **:zephyr_file:‘samples/bluetooth/central_hr‘** in the Zephyr tree.

See *bluetooth samples section* for details.

16.6.2 Bluetooth: Eddystone

Overview

Application demonstrating Eddystone Configuration Service

The Eddystone Configuration Service runs as a GATT service on the beacon while it is connectable and allows configuration of the advertised data, the broadcast power levels, and the advertising intervals. It also forms part of the definition of how Eddystone-EID beacons are configured and registered with a trusted resolver.

Requirements

- BlueZ running on the host, or
- A board with BLE support

Building and Running

This sample can be found under **:zephyr_file:‘samples/bluetooth/eddystone‘** in the Zephyr tree.

See *bluetooth samples section* for details.

16.6.3 Bluetooth: Peripheral_cts

Overview

Application demonstrating reading the time from a CTS service.

I used bluez on linux + a gatt server script that presents a current time service.

This program tests if it can get the time of a cts service. In order to get the time, you will have to connect the device first.

Requirements

- a bluetoothdevice running the CTS service
- a board with this software (pinetime)

You will need a serial port to read the output of the "printk" messages.

Building and Running

16.6.4 Bluetooth: Peripheral

Overview

Application demonstrating the BLE Peripheral role. It has several well-known and vendor-specific GATT services that it exposes.

Requirements

- BlueZ running on the host, or
- A board with BLE support

Building and Running

This sample can be found under **:pinetime_file:'samples/bluetooth/peripheral'** in the Zephyr tree.

See *bluetooth samples section* for details.

CHAPTER

SEVENTEEN

MENUCONFIG

17.1 Zephyr is like linux

TIP: the pinetime specific drivers are located under Modules

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  --->
```

(continues on next page)

(continued from previous page)

```
    Interrupt Controllers  --->
    Timer Drivers  --->
-*- 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

EIGHTEEN

HACKING STUFF

18.1 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
```

(continues on next page)

(continued from previous page)

```
wrote 24576 bytes from file zephyr.bin in 1.703540s (14.088 KiB/s)
** Programming Finished **

And finally execute a reset :
>reset
```

removing write protection see: *howto flash your zephyr image*

18.2 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
```

18.3 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.
```

18.3.1 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. . . .)

18.4 howto flash your zephyr image

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

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



I use Openocd to flash.
Just connect : telnet 127.0.0.1 4444

.. code-block:: console

    program zephyr.bin
```

18.5 howto remove the write protection

:: the PineTime watch is read/write protected (at least the one I got) executing the following : nrf52.dap apreg 1 0x0c shows 0x0

Mind you, st-link does not allow you to execute that command, for this you will need a J-link.

There is a workaround using the GPIO of a raspberry pi or an Orangepi. (in this case you won't need an external programmer at all) (You can find an example for the orange pi in my repo :https://github.com/najnesnaj/openocd.) You have to reconfigure Openocd with the –enable-cmsis-dap option.

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

18.6 howto configure gateway

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

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

```
I use an orange pi single board computer.
The pinetime watch is attached to this.
My development is done on a laptop.
```

```
How can you copy from one environment (laptop) to another (SBC) without typing␣
↪password?
```

On the laptop :

```
ssh-keygen -b 8092 -t rsa -C "fota gw access key" -f ~/.ssh/orange
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):          (LEAVE EMPTY!)
Enter same passphrase again:
Your identification has been saved in /root/.ssh/orange.
Your public key has been saved in /root/.ssh/orange.pub.
The key fingerprint is:
SHA256:xCM5Fk1LAVjEWqrM6LKM8Y6+Y12ONt6eV8vDa/KdRUM fota gw access key
The key's randomart image is:
+---[RSA 8092]----+
|     ==++.       |
|B*B.o+. +ooo     |
+----[SHA256]-----+

(the standard port is 22 and not 9988 which is my custom port)

copy the certificate to the SBC (which name is orange in my case):
ssh-copy-id -p 9988 -i ~/.ssh/orange.pub root@orange


create config file :  ~/.ssh/config
      Host orange
      HostName orange
      User root
      Port 9988
      IdentityFile ~/.ssh/orange
```

```
Now you can copy without a password :
scp build/zephyr/zephyr.bin orange:/usr/src
```

w

18.7 howto use 2 openocd sessions

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

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

(continues on next page)

(continued from previous page)

```
I use Openocd to flash.
Just connect : telnet 127.0.0.1 4444

.. code-block:: console

    program zephyr.bin
```

18.7.1 Suppose you have 2 microcontrollers

Just connect : telnet 127.0.0.1 7777 for the second.

18.7.2 Howto setup a second openocd session on a different port?

In this case an ST-LINK/V2 an in-circuit debugger and programmer is used.

```
openocd -c 'telnet_port 7777' -c 'tcl_port 6667' -c 'gdb_port 3332' -s /usr/local/
↪share/openocd/scripts -f interface/stlink.cfg  -f target/nrf52.cfg
```

18.7.3 Howto use the GPIO header of a Single Board computer

This works really well, and does not require a seperate programmer.

```
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'
```

18.8 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

NINETEEN

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.
```

19.1 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 . . .