open source watch Documentation

Release 1.0.0

jj

Nov 23, 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 DS-D6 SMARTWATCH

```
this document describes the installation of zephyr RTOS on the ds-d6 smartwatch.


It should be applicable on other nordic nrf52832 based watches (id107hr plus ....).
```

```
the approach in this manual is to get quick results :
    - minimal effort install
    - 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

https://docs.zephyrproject.org/latest/getting_started/index.html

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

In order to use the Nordic specific samples, you will need to install their environment, which can co-exist peacefully. The changes (updates) lag a month behind the community branch. https://github.com/nrfconnect/sdk-nrf

you can switch between these environments, by executing in their own directory

```
$ west update
```

CHAPTER

FIVE

ON THE SUBJECT OF UART

As I am predisposed of finding out things the hard way, it took me a while to notice why the serial port was only partly functioning :

- I used a wrong pin on my dev-board (id107 HR watch)
- I used a USB - el cheapo - serial port

The nrf52 is an arm chip with TTL levels of 3.3V, so you need a plug that can handle this voltage instead of the usual 5V.

I did not want to spend money and wait for the postman. I had an stm32f103c8 lying around.

This could be converted in a blackmagicprobe, which has a debugprobe and(!) a serial port. The stm32 is 3.3V compatible. Problem solved, which is quite cool, since it uses up only one USB-port.

5.1 read and learn

```
To compile a program with a non standard (=jlink) debugger:
west flash --runner blackmagicprobe
```

CHAPTER

SIX

THE BLACK MAGIC PROBE

6.1 probes in zephyr

You can program the nrf52832 with a debuggerprobe. The standard-setup is jlink (segger).

/root/zephyrproject/zephyr/boards/arm/id107plus/board.cmake (adapt the runner here)

in our case : instead of jlink specify : blackmagicprobe

The cool thing about this probe that it has a serial port (3.3V) and a debug (upload) port on the same usb-port.

- /dev/ttyACM1 is serial port (pb6 pb7)

minicom -b 115200 -D /dev/ttyACM1

- /dev/ttyACM0 is used as debugger/uploading

west debug –runner blackmagicprobe west flash –runner blackmagicprobe

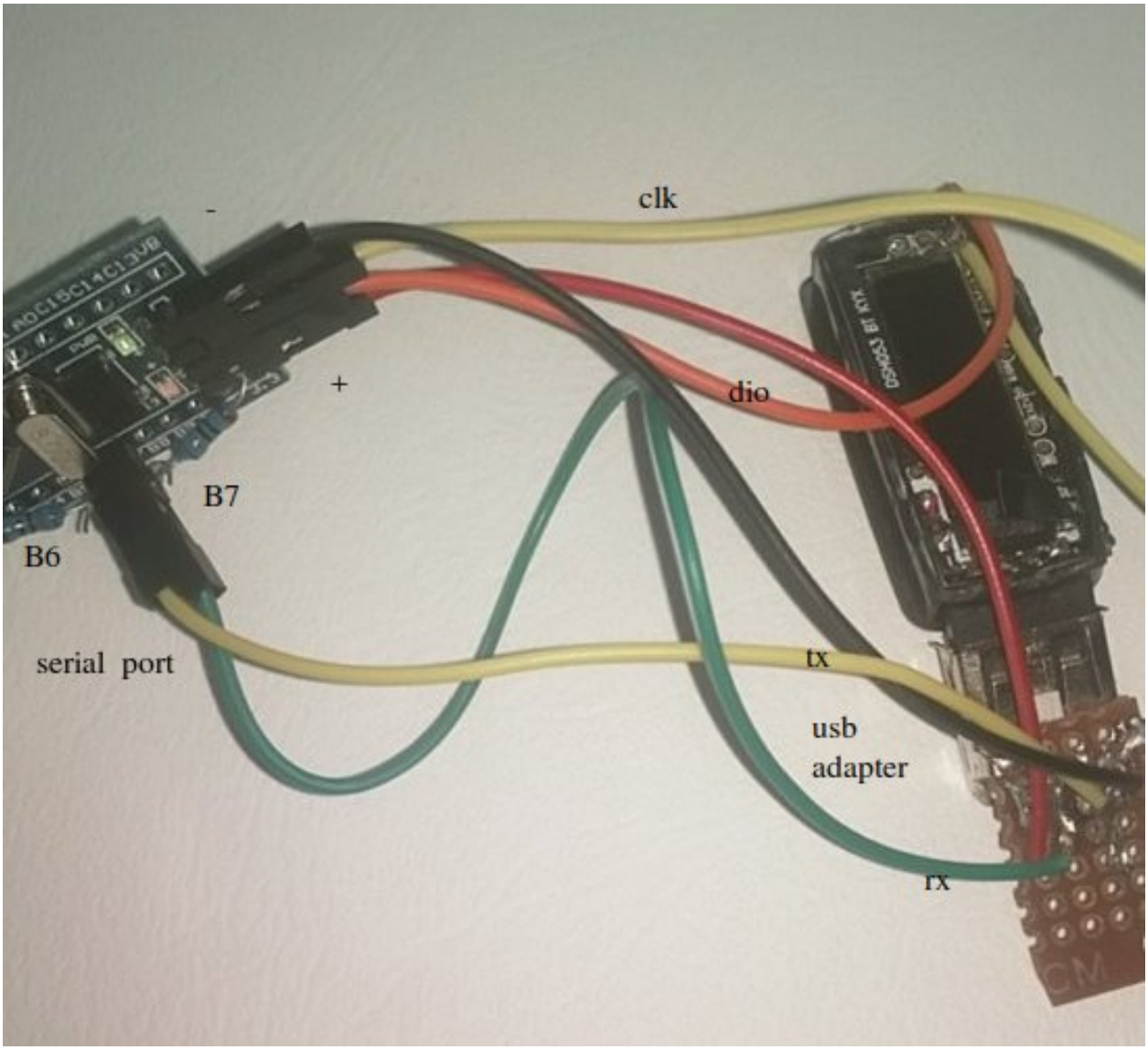
6.2 howto setup a blackmagicprobe

You can buy this probe and support the developers. (make this world a better place)

I bought a “cheapo” “blue pill” stm32 board for future projects . . . soldered a 1.8K resistor between 3.3K and PA12

downloaded from https://jeelabs.org/docs/software/bmp/ - blackmagic.bin (79 ko) - blackmagic_dfu.bin (7 ko)

in jlink : loadbin blackmagic_dfu.bin 0x8000000 (specify jlink no options . . .) switch boot0 or boot1 or whatever connect usb in linux dfu-util -v -R -d 0483:df11 -s 0x08002000 -D blackmagic.bin (uploading in jlink was a problem cause memory restrictions)



(removed boot0 and boot1 connectors on the stm afterwards)

plugged it in the USB port and it pops up (had to enable it first in virtual box usb : black sphere technologies . . . . .)

CHAPTER

SEVEN

STARTING WITH SOME BASIC APPLICATIONS

The best way to get a feel of zephyr, is to start building sample 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.

7.1 Blinky example

```
The watch does not contain a led as such, but it has a heart rate sensor.

Powering the heart rate sensor, lights up the green led.
```

have a look at the ds_d6.dts file, here you see the definition of the led.

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

see : blinky-sample

```
$ cd ~/zephyrproject/zephyr
$ west build -p -b ds_d6 samples/basic/blinky
```

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

you can upload it with:

west flash or west flash –runner blackmagicprobe (or jlink, or ....)

CHAPTER

EIGHT

BLUETOOTH (BLE) EXAMPLE

The ds-d6 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.

8.1 Using the created bluetooth sample:

I use linux with a bluetoothadapter 4.0. You will 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.

8.2 Ble Peripheral

this example is a demo of the services under bluetooth

first build the image

```
$  west build -p -b ds_d6 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

8.3 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

NINE

DISPLAY (SSD1306)

in order to get the ds d6 display working you will need to replace the standard display driver by the one provided in the directory drivers (file ssd1306.c)

for the id107 hr plus you will need the ssd1306.c-id107hrplus.

9.1 Display example

The display uses the character frame buffer (cfb).

There are two samples which should work out of the box.

check out the cfb_shell - samples, which is really cool. You will need a serial connection :eg. : minicom -b 115200 -D /dev/ttyACM1. (just type help to get an overview of commands)

```
$  cfb init
$  cfb print 0 0 "hello world"
```

```
$  west build -p -b ds_d6 samples/display/cfb
or
$  west build -p -b ds_d6 samples/display/cfb_shell
```

Displaying graphics should possible with the lvgl module. (work in progress)

CHAPTER

TEN

LITTLEVGL BASIC SAMPLE

10.1 Overview

This sample application displays “Hello World” 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 Todo

- This is work in progress. It should be possible, but since OLED is monochrome, it will needs some tweeking.

10.3 References

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

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

CHAPTER

ELEVEN

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

11.1 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)

11.2 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 ds_d6 samples/bluetooth/peripheral-cts
```

11.3 Using bluez on linux to connect

The ds_d6 zephyr sample behaves as a peripheral:

- first of all start the cts service

  -connect to the ds_d6 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)
```

11.4 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
```

11.5 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

TWELVE

FIRMWARE OVER THE AIR (FOTA)

12.1 Serial Device Firmware Upgrade

My main aim is to get this working. It has a small footprint, since no need for bluetooth.

I have not(!) been successfull.

west build -b ds_d6 samples/display/cfb – -DOVERLAY_CONFIG='overlay-serial.conf;overlay-fs.conf' -DCONFIG_MCUBOOT_SIGNATURE_KEY_FILE="../bootloader/mcuboot/root-rsa-2048.pem"

12.2 Wireless Device Firmware Upgrade

12.2.1 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)

12.3 MCUboot with zephyr

west build -b ds_d6 -s ../bootloader/mcuboot/boot/zephyr -d build-mcuboot west flash -d build-mcuboot –runner blackmagicprobe

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.

west build -b ds_d6 samples/subsys/console/echo -d build-hello-signed -DCONFIG_MCUBOOT_SIGNATURE_KEY_FILE="../bootloa
rsa-2048.pem" west flash -d build-hello-signed –runner blackmagicprobe

another example with overlays: west build -b ds_d6 samples/subsys/mgmt/mcumgr/smp_svr – -DOVERLAY_CONFIG='overlay-serial.conf;overlay-fs.conf'

mcumgr -ldebug -t 60 –conntype=serial –connstring='dev=/dev/ttyACM1,baud=115200' image list

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.

west flash knows where to put the application (you do not have to specify the address of the slot)

12.4 Partitions

```
have a look at boards/ds_d6.dts
```

12.4.1 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

12.4.2 Remark

the ds_d6 has no nor flash. Having 2 slots limits the size of you program! You can also use 1 slot. (is less secure of course)

12.5 Signing an application

Never use the default public key, but generate your own!

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

12.5.1 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
```

12.5.2 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.

12.5.3 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
```

12.6 SMP Server Sample

12.6.1 Overview

This sample application implements a Simple Management Protocol (SMP) server. SMP is a basic transfer encoding for use with the MCUmgr management protocol.

This sample application supports the following mcumgr transports by default:

- Shell
- Bluetooth

12.6.2 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)

It is written in the go-language. You need to adapt the path : PATH=$PATH:/root/go/bin.

12.6.3 Building and Running

The sample will let you manage the pinetime over bluetooth. (via SMP protocol)

There are slot0 and slot1 which can both contain firmware.

Suppose you switch from slot0 to slot1, you still want to be able to communicate.

So both slots need smp_svr software!

Step 1: Build smp_svr

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

NOTE: to perform a firmware update over the air, you have to build a second sample

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 pinetime, 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
* Sign the second sample
* 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.

You can 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

THIRTEEN

SAMPLES AND DEMOS

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

13.1 Character frame buffer

13.1.1 Overview

This sample displays character strings using the Character Frame Buffer (CFB) subsystem framework.

13.1.2 Building and Running

build the application: west build -p -b ds_d6 samples/display/cfb

on unix : #minicom -b 115200 -D /dev/ttyACM1

you get a shell and you can type help

to display something on the screen : cfb init cfb invert cfp print 0 0 “hello world”

13.2 Bluetooth: Peripheral UART

for this you will need the nordic branch of zephyr : nrfconnect/sdk-zephyr

13.3 TODO

this samples prints data on serial port on the bluetooth service (and the other way around)

it would be cool if you could use cfb (character frame buffer) as well. (send messages to screen)

The Peripheral UART sample demonstrates how to use the nus_service_readme. It uses the NUS service to send data back and forth between a UART connection and a Bluetooth LE connection, emulating a serial port over Bluetooth LE.

13.3.1 Overview

When connected, the sample forwards any data received on the RX pin of the UART 1 peripheral to the Bluetooth LE unit. On Nordic Semiconductor's development kits, the UART 1 peripheral is typically gated through the SEGGER chip to a USB CDC virtual serial port.

Any data sent from the Bluetooth LE unit is sent out of the UART 1 peripheral's TX pin.

13.3.2 Debugging

In this sample, the UART console is used to send and read data over the NUS service. Debug messages are not displayed in this UART console. Instead, they are printed by the RTT logger.

If you want to view the debug messages, follow the procedure in testing_rtt_connect.

13.3.3 Building and running

Testing

13.3.4 Dependencies

This sample uses the following |NCS| libraries:

- nus_service_readme

In addition, it uses the following Zephyr libraries:

- `include/zephyr/types.h`
- `boards/arm/nrf*/board.h`
- zephyr:kernel_api:
  - `include/kernel.h`
- zephyr:api_peripherals:
  - `incude/gpio.h`
  - `include/uart.h`
- zephyr:bluetooth_api:
  - `include/bluetooth/bluetooth.h`
  - `include/bluetooth/gatt.h`
  - `include/bluetooth/hci.h`
  - `include/bluetooth/uuid.h`

13.4 Bluetooth: NUS shell transport

for this, you will need the nordic branch of zephyr (https://github.com/nrfconnect/sdk-zephyr)

The Nordic UART Service (NUS) shell transport sample demonstrates how to use the shell_bt_nus_readme to receive shell commands from a remote device.

13.5 TODO

You can use the shell to type messages on the OLED screen (cfb character frame buffer) You can use bluetooth shell to type messages on OLED screen.

It would be really cool to use bt-shell to type message on uart (and the other way around)

I experimented with bt-shell and here you can use gatt to type characters on bluetooth.

13.5.1 Overview

When the connection is established, you can connect to the sample through the nus_service_readme by using a host application. You can then send shell commands, that are executed on the device that runs the sample, and see the logs. See shell_bt_nus_host_tools for more information about the host tools available, in |NCS|, for communicating with the sample.

13.5.2 Building and running

Testing using the BLE Console

See ble_console_readme for more information on how to test the sample using the BLE Console.

13.5.3 Dependencies

This sample uses the following |NCS| libraries:

- shell_bt_nus_readme
- nus_service_readme

In addition, it uses the following Zephyr libraries:

- zephyr:bluetooth_api:
  - `include/bluetooth/bluetooth.h`
  - `include/bluetooth/hci.h`
  - `include/bluetooth/uuid.h`
  - `include/bluetooth/gatt.h`
  - `samples/bluetooth/gatt/bas.h`
- zephyr:logging_api

CHAPTER

FOURTEEN

MENUCONFIG

14.1 Zephyr is like linux

Once you have build a sample, you can add extra stuff by using the configuration menu.

```
$ 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  --->
    C Library  --->
    Additional libraries  --->
[*] Bluetooth  --->
[ ] C++ support for the application  ----
    System Monitoring Options  --->
    Debugging Options  --->
[ ] Disk Interface  ----
    File Systems  --->
-*- Logging  --->
    Management  --->
    Networking  --->
```

14.1.1 after saving :

you will need to rebuild the image to include changes in config

```
$ west build
```

CHAPTER

FIFTEEN

HACKING STUFF

15.1 debugging the ds-d6 smartwatch

```
The ds-D6 has a serial port.
```

The blackmagicprobe can lauch a debugger : west debug –runnerblackmagicprobe. The probe has a serial port /dev/ttyACM1 (linux : minicom -b 115200 -D /dev/ttyACM1)

The Segger Jlink, which is kind of standard in zephyr can use the the swd-connector for debug messages as well.

Note: The Segger JLink offers the JLinkRTTViewer. In order to use it, you can set this in ds_d6_defconfig (boardconfig) CONFIG_LOG=y CONFIG_USE_SEGGER_RTT=y CONFIG_SHELL=y CONFIG_SHELL_BACKEND_RTT=y

sniffing memory

```
The JLink probe allowed to check memory at 0x5000504 and 0x50000514.
#mem32 0x5000504 0x1
or write a value
#w4 0x50000504 0x12345678
This allowed me to check GPIO ports.
```

15.2 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

SIXTEEN

ABOUT

The Desay D6 smart watch, contains the nordic nrf52832 microcontroller.

This watch has an serial port.

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.
```

16.1 What is already working :

- wireless hack : one can install espruino wirelessly (github.com/fanoush/ds-d6)
- zephyr RTOS with working screen