PULP PLATFORM

Open Source Hardware, the way it should be!

Bitcraze Workshop: AI-deck

The Application Layer

Lorenzo Lamberti, Hanna Müller, *Vlad Niculescu*, Manuele Rusci, Daniele Palossi

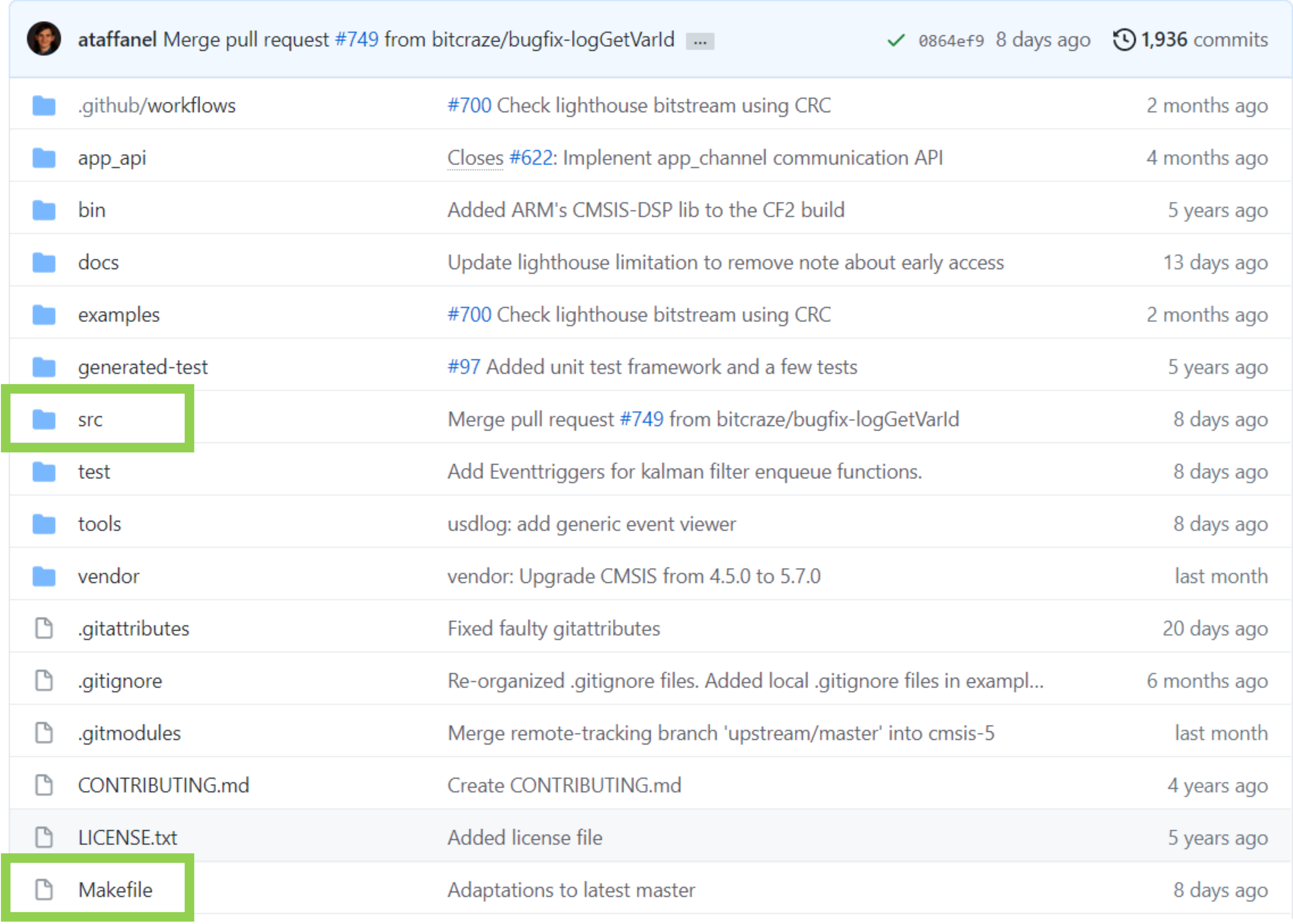
http://pulp-platform.org @pulp_platform https://www.youtube.com/pulp_platform

Firmware Overview

- Open-source, available at: https://github.com/bitcraze/crazyflie-firmware.
- Based on FreeRTOS.
- The firmware implements solutions for: state estimation, control, logging, trajectory planning, etc.
- It implements the sensor drivers and deck drivers.
  Deck: a plug-in PCB that is attached to the Crazyflie.
- The user can add new functionalities.

Firmware Overview

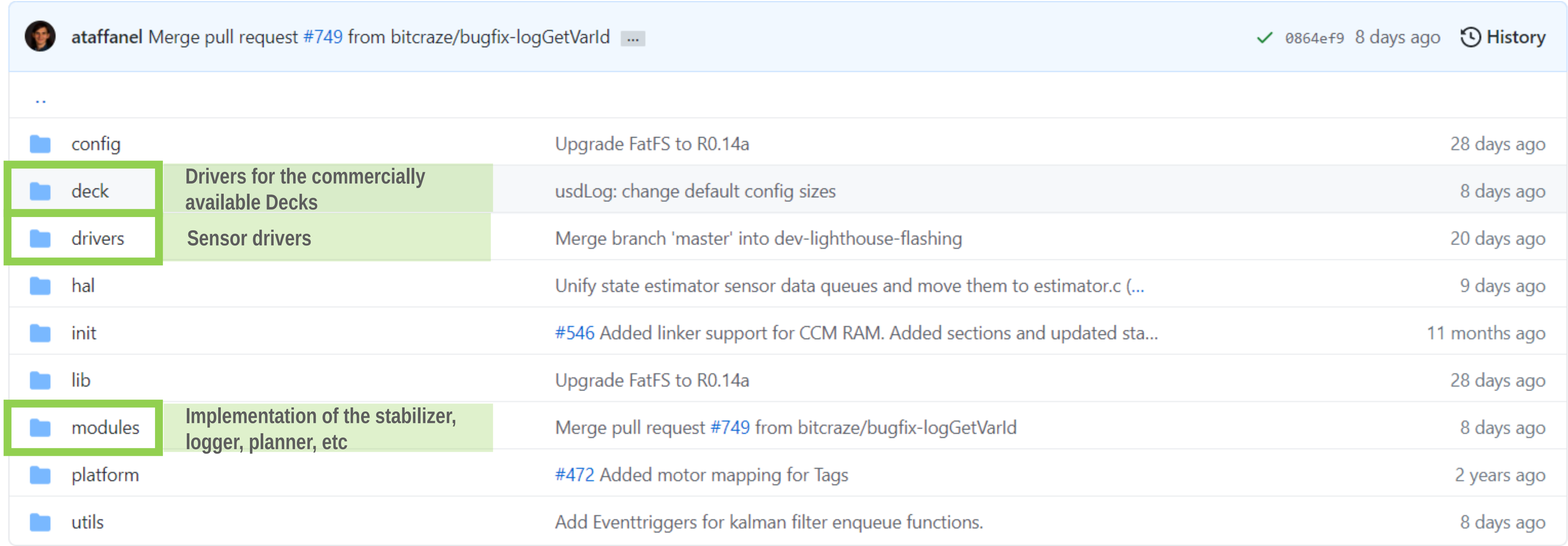
Firmware source files



ataffanel Merge pull request #749 from bitcraze/bugfix-logGetVarId ... ✓ 0864ef9 8 days ago 1,936 commits

.github/workflows	#700 Check lighthouse bitstream using CRC	2 months ago
app_api	Closes #622: Implenent app_channel communication API	4 months ago
bin	Added ARM's CMSIS-DSP lib to the CF2 build	5 years ago
docs	Update lighthouse limitation to remove note about early access	13 days ago
examples	#700 Check lighthouse bitstream using CRC	2 months ago
generated-test	#97 Added unit test framework and a few tests	5 years ago
src	Merge pull request #749 from bitcraze/bugfix-logGetVarId	8 days ago
test	Add Eventtriggers for kalman filter enqueue functions.	8 days ago
tools	usdlog: add generic event viewer	8 days ago
vendor	vendor: Upgrade CMSIS from 4.5.0 to 5.7.0	last month
.gitattributes	Fixed faulty gitattributes	20 days ago
.gitignore	Re-organized .gitignore files. Added local .gitignore files in exampl...	6 months ago
.gitmodules	Merge remote-tracking branch 'upstream/master' into cmsis-5	last month
CONTRIBUTING.md	Create CONTRIBUTING.md	4 years ago
LICENSE.txt	Added license file	5 years ago
Makefile	Adaptations to latest master	8 days ago

Firmware Overview – Source Files



ataffanel Merge pull request #749 from bitcraze/bugfix-logGetVarId ... ✓ 0864ef9 8 days ago History

..

config	Upgrade FatFS to R0.14a	28 days ago
deck	usdLog: change default config sizes	8 days ago
drivers	Merge branch 'master' into dev-lighthouse-flashing	20 days ago
hal	Unify state estimator sensor data queues and move them to estimator.c (...	9 days ago
init	#546 Added linker support for CCM RAM. Added sections and updated sta...	11 months ago
lib	Upgrade FatFS to R0.14a	28 days ago
modules	Merge pull request #749 from bitcraze/bugfix-logGetVarId	8 days ago
platform	#472 Added motor mapping for Tags	2 years ago
utils	Add Eventtriggers for kalman filter enqueue functions.	8 days ago

deck: Drivers for the commercially available Decks

drivers: Sensor drivers

modules: Implementation of the stabilizer, logger, planner, etc

Developping Your Own Application

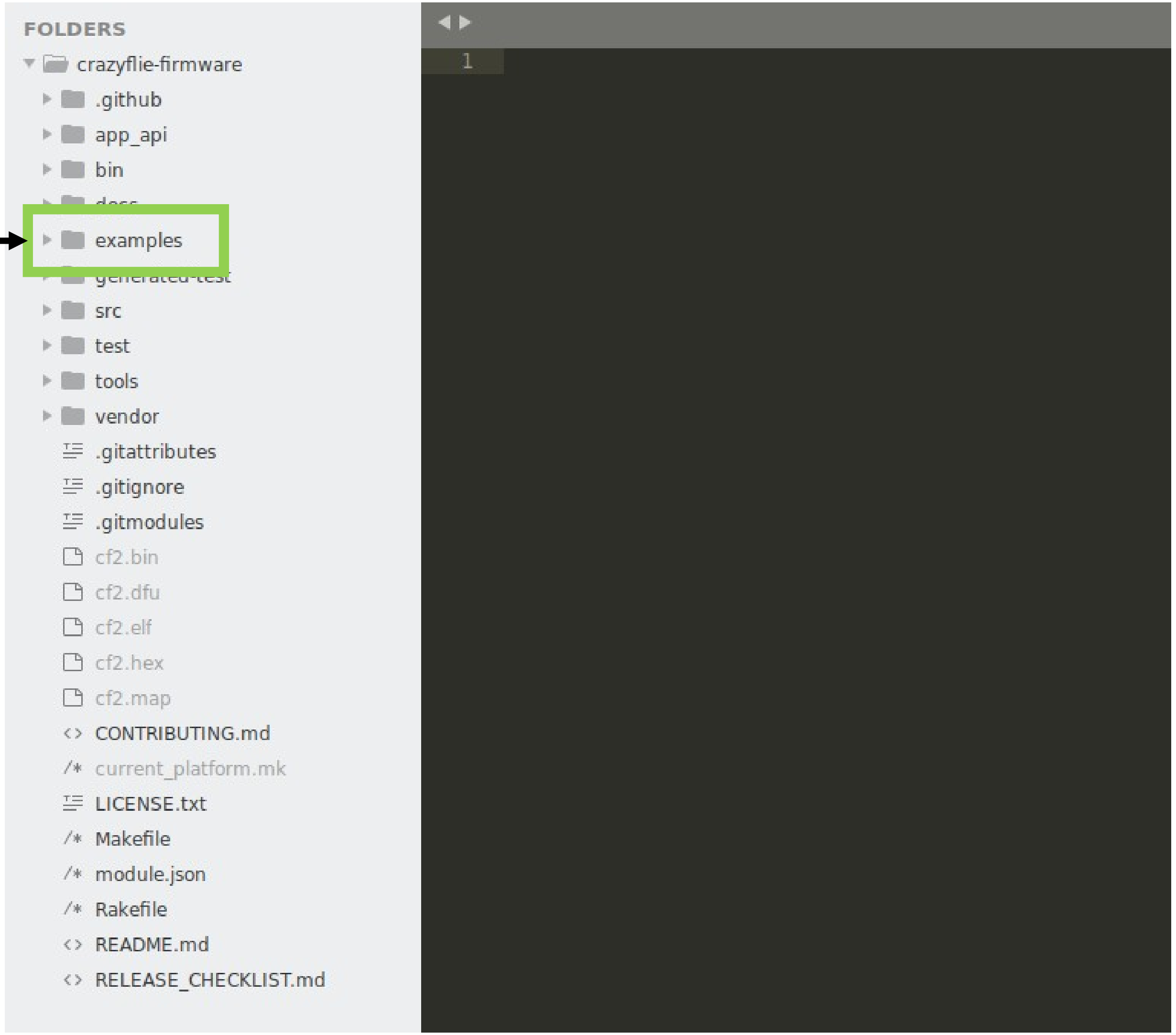
- One option for developing with Crazyflie, is to add the new source files to the *modules* or as a new *deck*.
- Not the best practice, since it alters the firmware and could cause conflicts with future updates (i.e., git pull conflicts).

Developping Your Own Application

- The *Application Layer* feature of the firmware allows the user to develop an application without changing the firmware.
- The code written within an application, is integrated as a new task and executed by the scheduler of the main firmware.

Firmware Overview

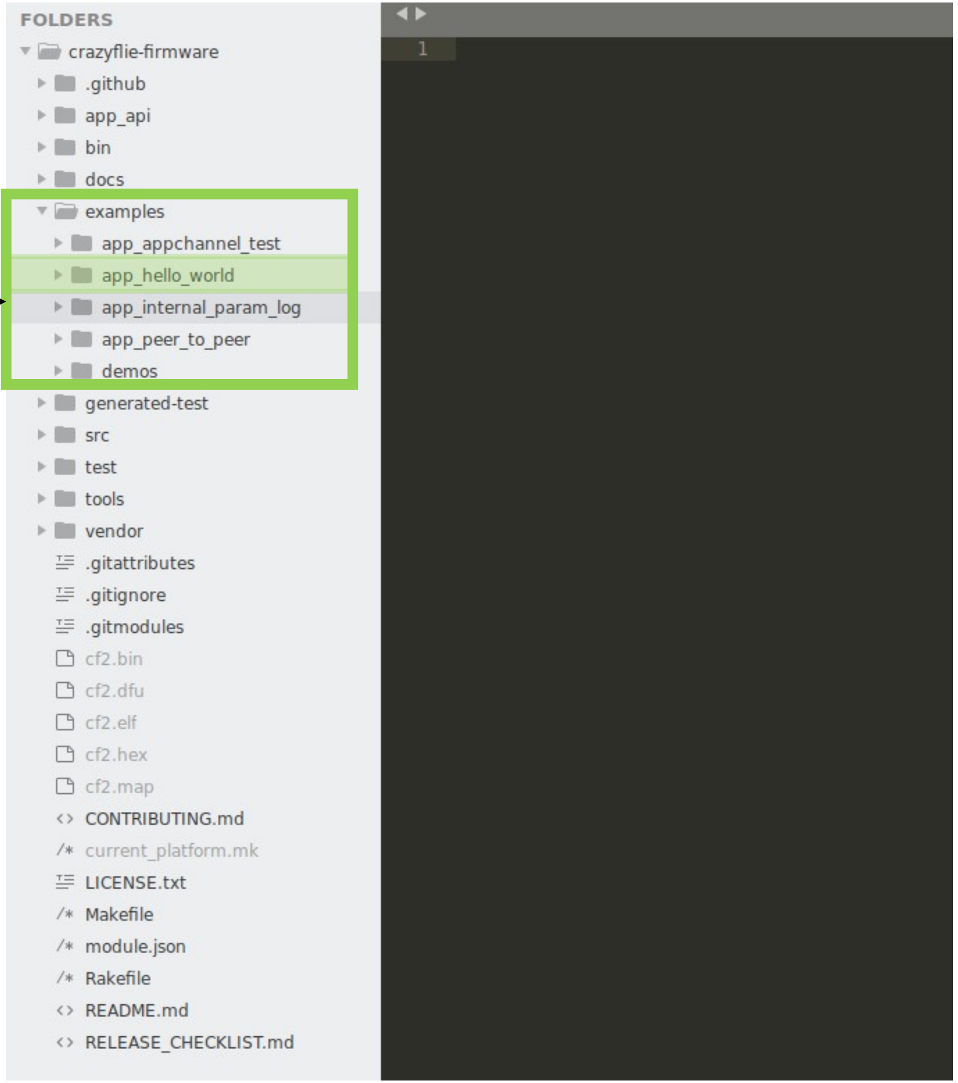
Examples on developing using the Application Layer



```
FOLDERS
▼ crazyflie-firmware
  ▶ .github
  ▶ app_api
  ▶ bin
  ▶ docs
  ▶ examples
  ▶ generated-test
  ▶ src
  ▶ test
  ▶ tools
  ▶ vendor
    .gitattributes
    .gitignore
    .gitmodules
    cf2.bin
    cf2.dfu
    cf2.elf
    cf2.hex
    cf2.map
    CONTRIBUTING.md
    current_platform.mk
    LICENSE.txt
    Makefile
    module.json
    Rakefile
    README.md
    RELEASE_CHECKLIST.md
```

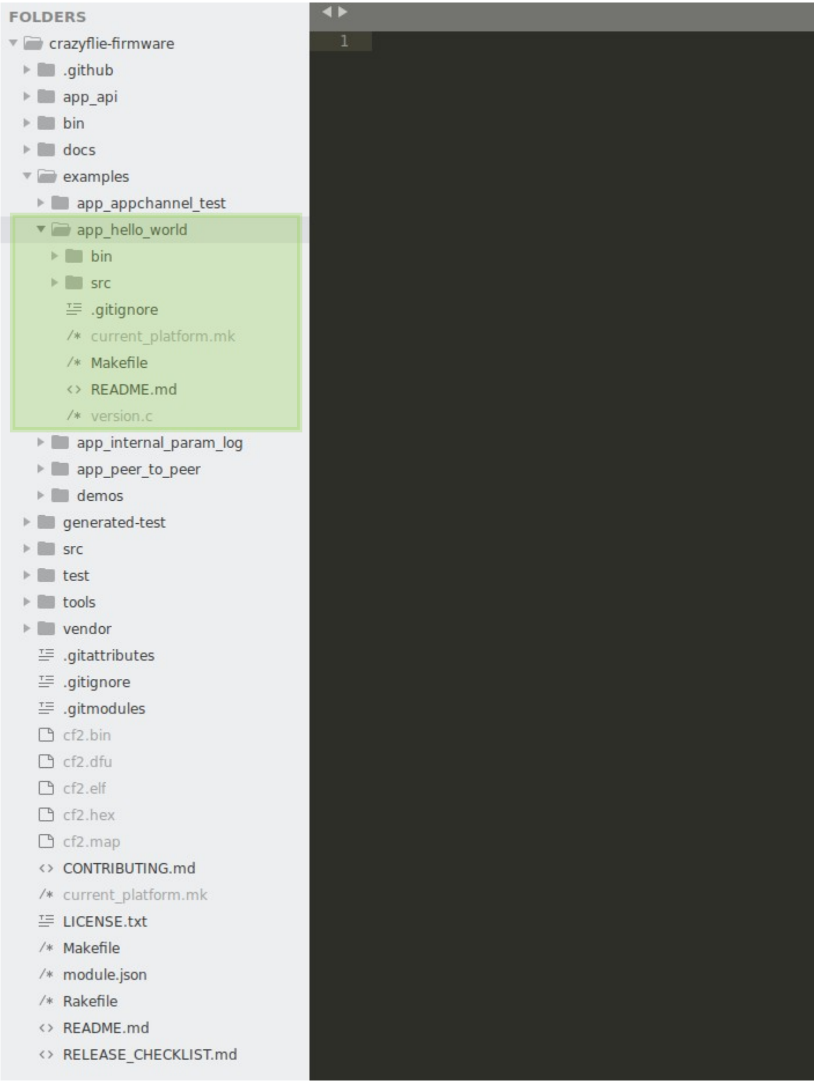
Example Applications

Examples on developing using the Application Layer



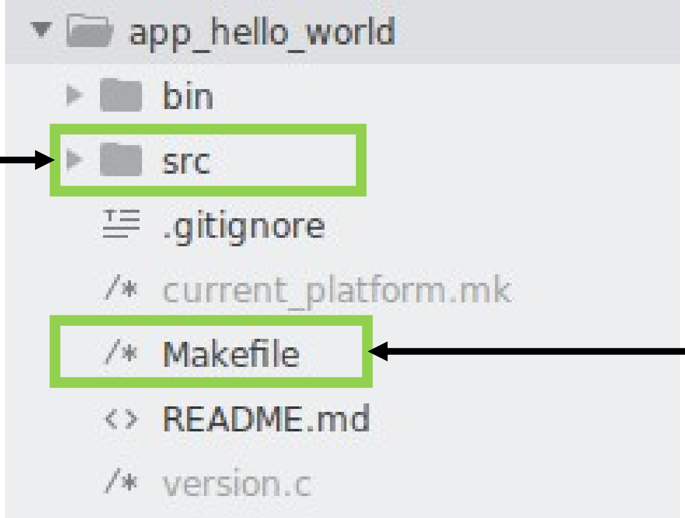
FOLDERS
- crazyflie-firmware
  - .github
  - app_api
  - bin
  - docs
  - examples
    - app_appchannel_test
    - app_hello_world
    - app_internal_param_log
    - app_peer_to_peer
    - demos
  - generated-test
  - src
  - test
  - tools
  - vendor
  - .gitattributes
  - .gitignore
  - .gitmodules
  - cf2.bin
  - cf2.dfu
  - cf2.elf
  - cf2.hex
  - cf2.map
  - CONTRIBUTING.md
  - current_platform.mk
  - LICENSE.txt
  - Makefile
  - module.json
  - Rakefile
  - README.md
  - RELEASE_CHECKLIST.md

Example Applications



```
FOLDERS
▾ crazyflie-firmware
  ▸ .github
  ▸ app_api
  ▸ bin
  ▸ docs
  ▾ examples
    ▸ app_appchannel_test
    ▾ app_hello_world
      ▸ bin
      ▸ src
        .gitignore
        current_platform.mk
        Makefile
        README.md
        version.c
    ▸ app_internal_param_log
    ▸ app_peer_to_peer
    ▸ demos
  ▸ generated-test
  ▸ src
  ▸ test
  ▸ tools
  ▸ vendor
    .gitattributes
    .gitignore
    .gitmodules
    cf2.bin
    cf2.dfu
    cf2.elf
    cf2.hex
    cf2.map
    CONTRIBUTING.md
    current_platform.mk
    LICENSE.txt
    Makefile
    module.json
    Rakefile
    README.md
    RELEASE_CHECKLIST.md
```

Example Application – Hello World

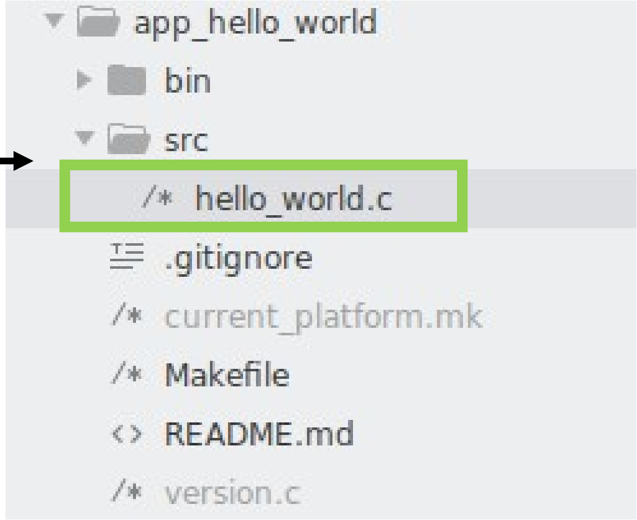


```
app_hello_world
  bin
  src
  .gitignore
  current_platform.mk
  Makefile
  README.md
  version.c
```

Project source files. Contains the new code developed by the user.

Project's Makefile. It is appended to the firmware's Makefile. At compilation time, both the firmware and the application get compiled.

Example Application – Hello World

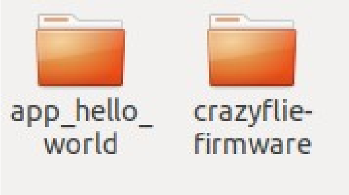


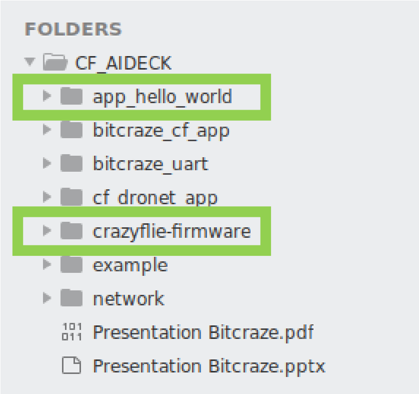
```
app_hello_world
  bin
  src
    hello_world.c
  .gitignore
  current_platform.mk
  Makefile
  README.md
  version.c
```

Source file that contains the application's code

Moving the application outside the firmware

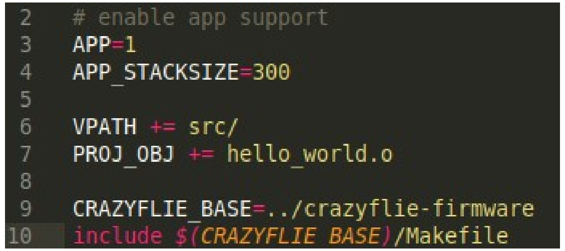
- The application code can be kept outside the main firmware.
- The *app_hello_world* project can be moved at the same level with the *crazyflie-firmware* folder.





```
FOLDERS
▾ CF_AIDECK
    ▸ app_hello_world
    ▸ bitcraze_cf_app
    ▸ bitcraze_uart
    ▸ cf_dronet_app
    ▸ crazyflie-firmware
    ▸ example
    ▸ network
      Presentation Bitcraze.pdf
      Presentation Bitcraze.pptx
```

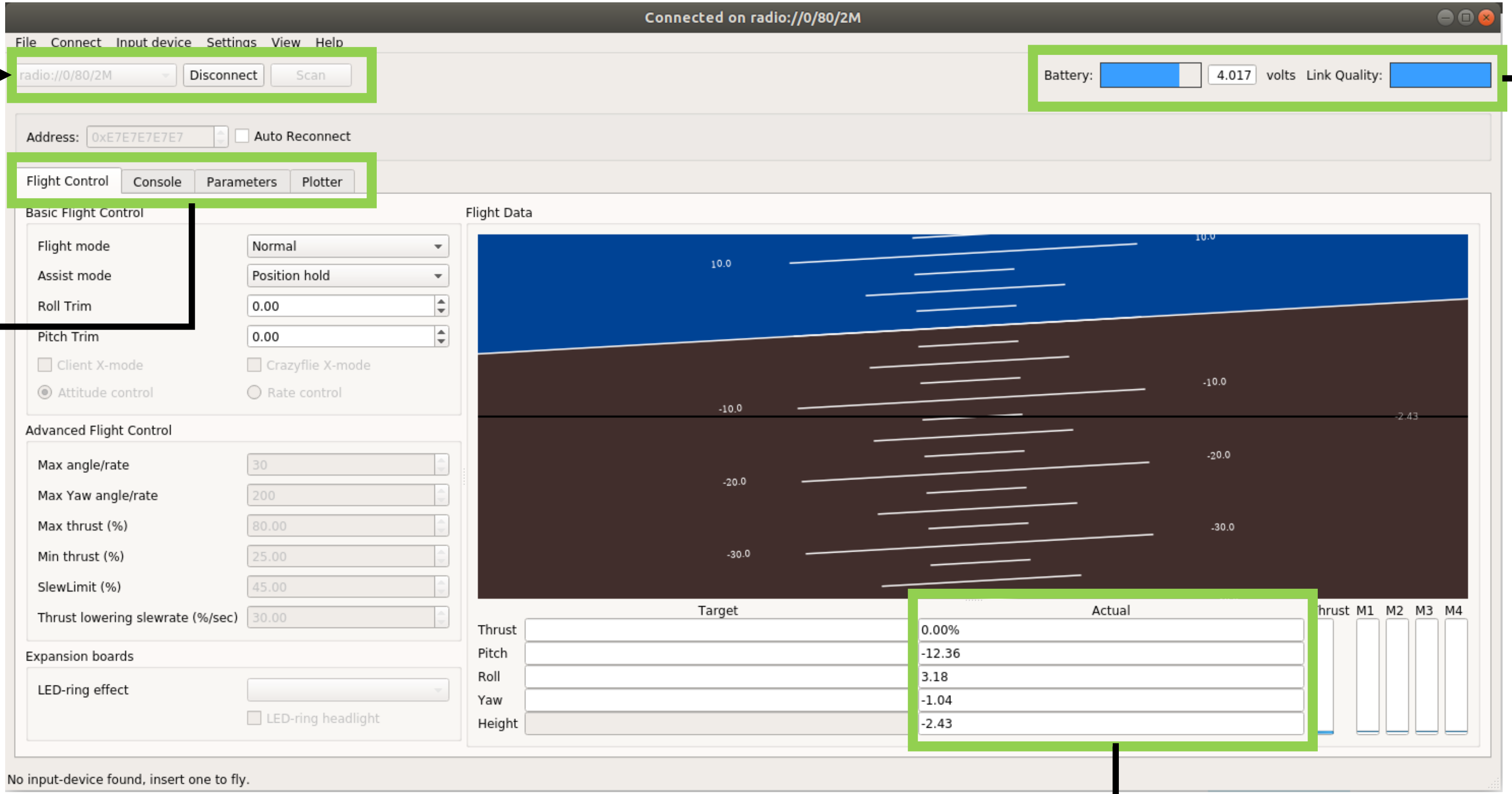
- It is required to inform the application where the firmware folder is located, by modifying its Makefile.



```
# enable app support
APP=1
APP_STACKSIZE=300

VPATH += src/
PROJ_OBJ += hello_world.o

CRAZYFLIE_BASE=../crazyflie-firmware
include $(CRAZYFLIE_BASE)/Makefile
```

The Crazyflie Client - Overview

- Allows the user to interact with the Crazyflie via USB or Radio



Connected on radio://0/80/2M

Battery: 4.017 volts Link Quality:

Address: 0xE7E7E7E7E7 □ Auto Reconnect

Basic Flight Control

Flight mode	Normal
Assist mode	Position hold
Roll Trim	0.00
Pitch Trim	0.00
□ Client X-mode	□ Crazyflie X-mode
● Attitude control	○ Rate control

Advanced Flight Control

Max angle/rate	30
Max Yaw angle/rate	200
Max thrust (%)	80.00
Min thrust (%)	25.00
SlewLimit (%)	45.00
Thrust lowering slewrate (%/sec)	30.00

Expansion boards

LED-ring effect

□ LED-ring headlight

Flight Data

	Target	Actual
Thrust		0.00%
Pitch		-12.36
Roll		3.18
Yaw		-1.04
Height		-2.43

No input-device found, insert one to fly.

Connect to the desired drone.

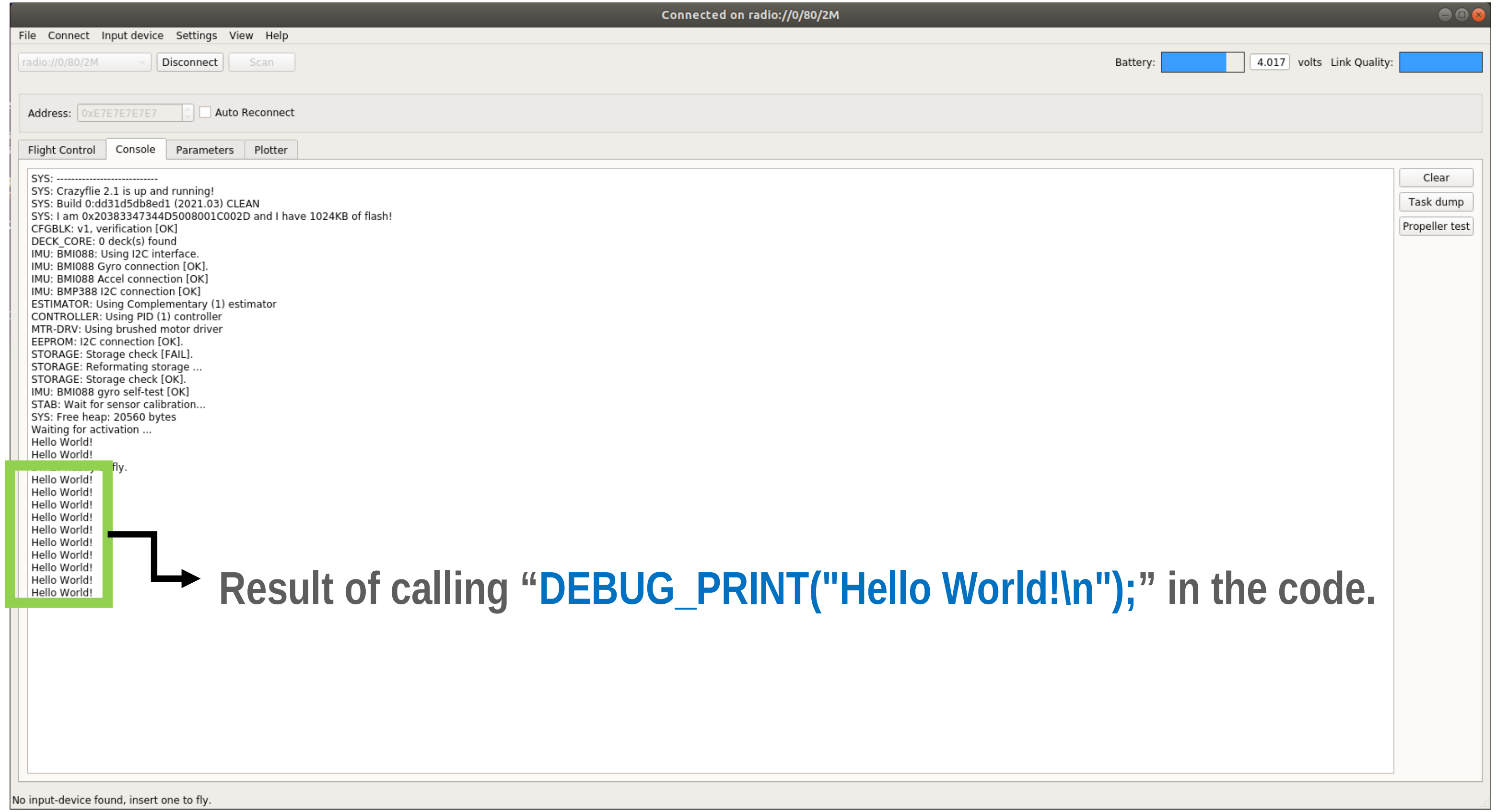
Each tab represent a functionality of the Client. More can be added via the View menu.

Check drone’s battery level and Crazyradio’s signal strength.

Observe the attitude

The Crazyflie Client - Console

- The console displays what is printed in the firmware via the DEBUG_PRINT function: strings and variables' values



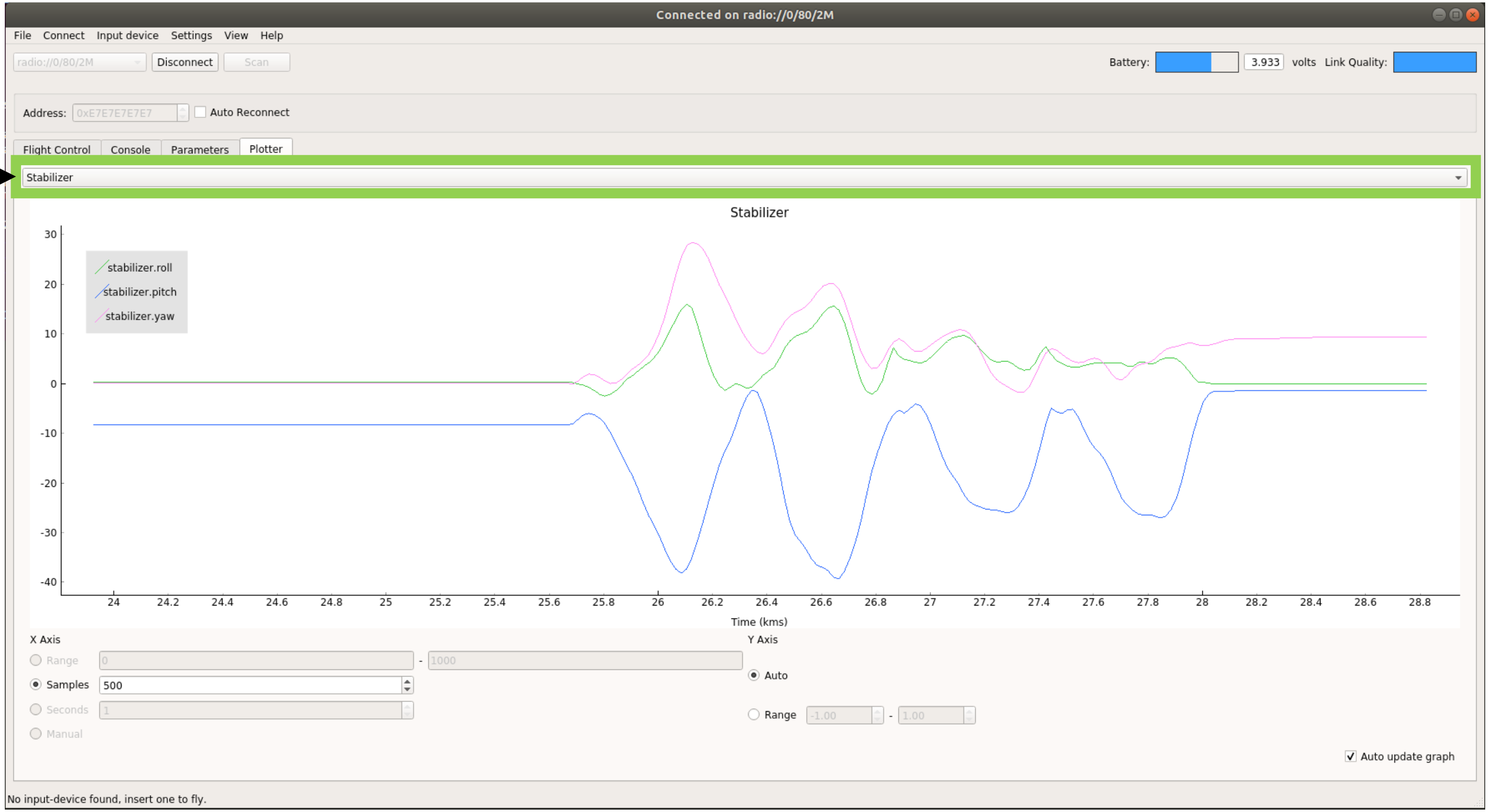
```
SYS: ----------------------------
SYS: Crazyflie 2.1 is up and running!
SYS: Build 0:dd31d5db8ed1 (2021.03) CLEAN
SYS: I am 0x20383347344D5008001C002D and I have 1024KB of flash!
CFGBLK: v1, verification [OK]
DECK_CORE: 0 deck(s) found
IMU: BMI088: Using I2C interface.
IMU: BMI088 Gyro connection [OK].
IMU: BMI088 Accel connection [OK]
IMU: BMP388 I2C connection [OK]
ESTIMATOR: Using Complementary (1) estimator
CONTROLLER: Using PID (1) controller
MTR-DRV: Using brushed motor driver
EEPROM: I2C connection [OK].
STORAGE: Storage check [FAIL].
STORAGE: Reformating storage ...
STORAGE: Storage check [OK].
IMU: BMI088 gyro self-test [OK]
STAB: Wait for sensor calibration...
SYS: Free heap: 20560 bytes
Waiting for activation ...
Hello World!
Hello World!
            fly.
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

No input-device found, insert one to fly.

Result of calling "DEBUG_PRINT("Hello World!\n");" in the code.

The Crazyflie Client - Plotter

- Allows plotting the logged variables and monitor their evolution in time.



Connected on radio://0/80/2M

Battery: 3.933 volts Link Quality:

Address: 0xE7E7E7E7E7 □ Auto Reconnect

Stabilizer

X Axis: ○ Range 0 - 1000; ● Samples 500; ○ Seconds 1; ○ Manual

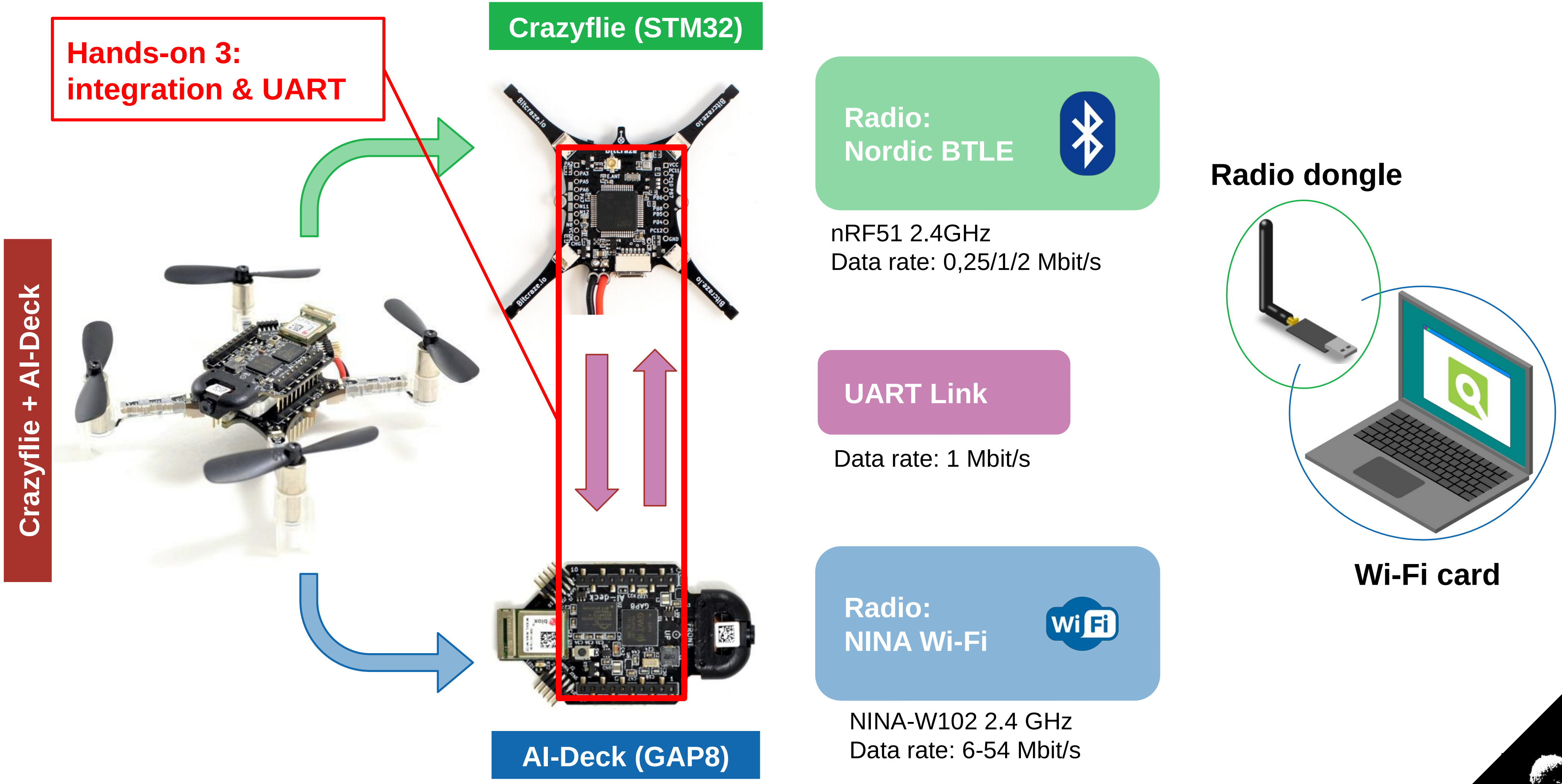
Y Axis: ● Auto; ○ Range -1.00 - 1.00

☑ Auto update graph

No input-device found, insert one to fly.

Select variables to plot.

The AI-Deck



Hands-on 3:
integration & UART

Crazyflie (STM32)

Crazyflie + AI-Deck

AI-Deck (GAP8)

Radio:
Nordic BTLE

nRF51 2.4GHz
Data rate: 0,25/1/2 Mbit/s

UART Link

Data rate: 1 Mbit/s

Radio:
NINA Wi-Fi

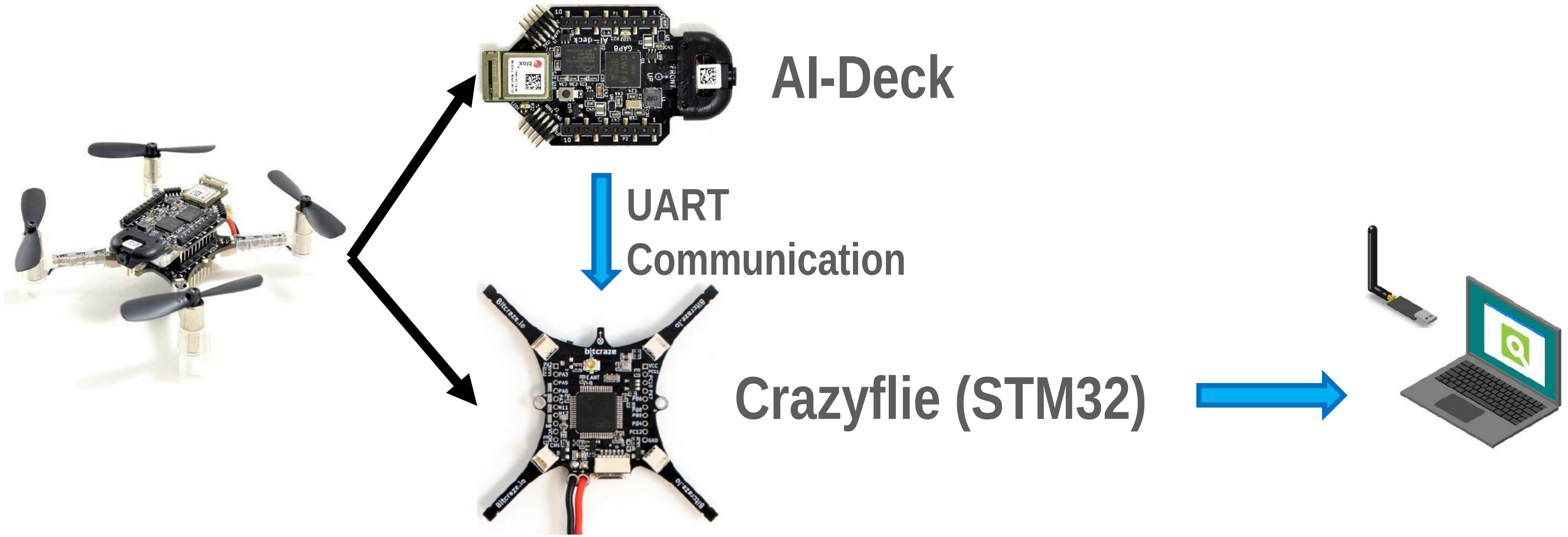
NINA-W102 2.4 GHz
Data rate: 6-54 Mbit/s

Radio dongle

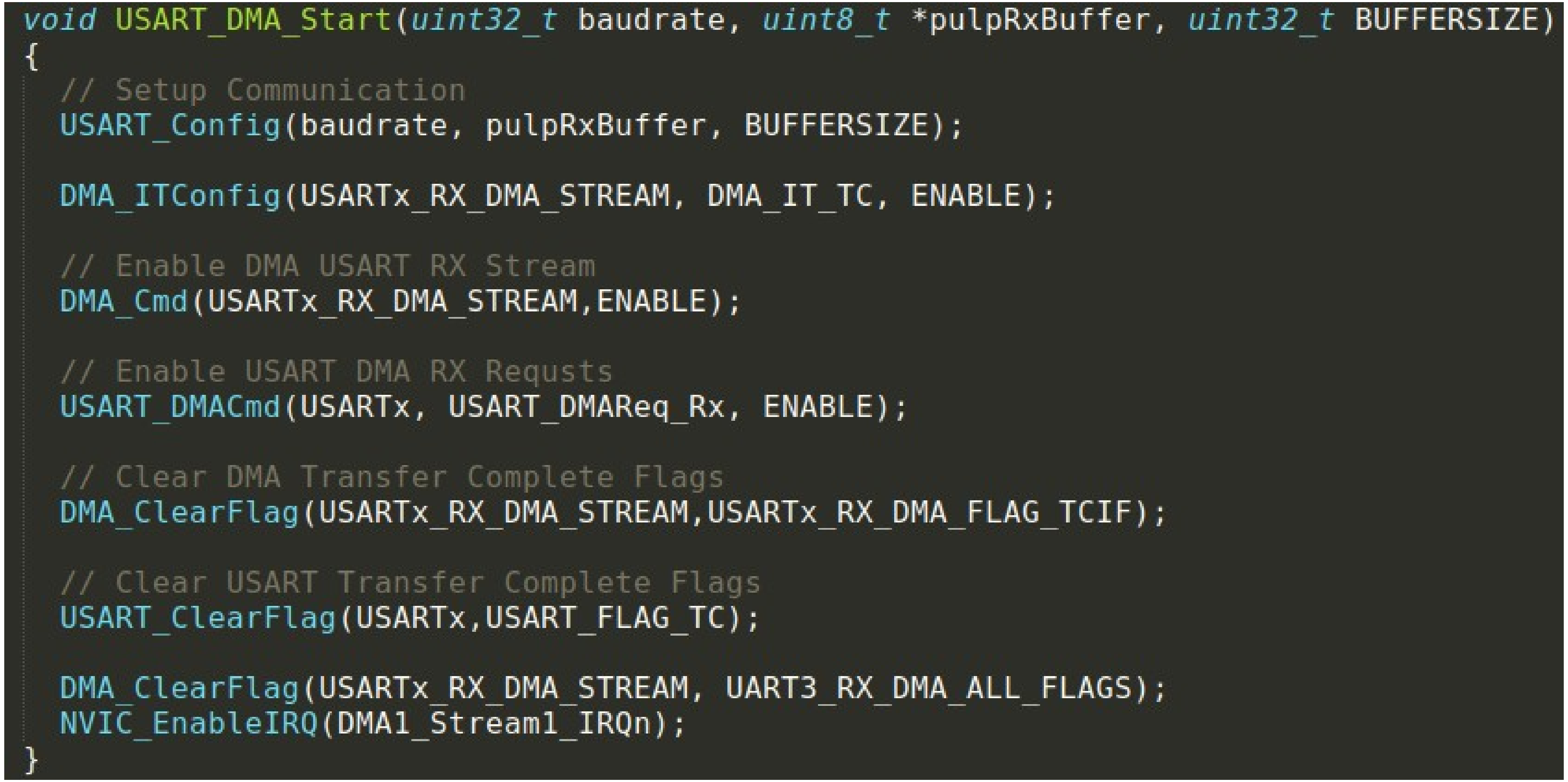
Wi-Fi card

Application Example

- Example: AI-Deck is sending the value of a counter every 0.5s.
- The Crazyflie prints every value that it receives.
- The Crazyflie uses the UART with DMA, which triggers an interrupt whenever a certain amount of bytes was received.



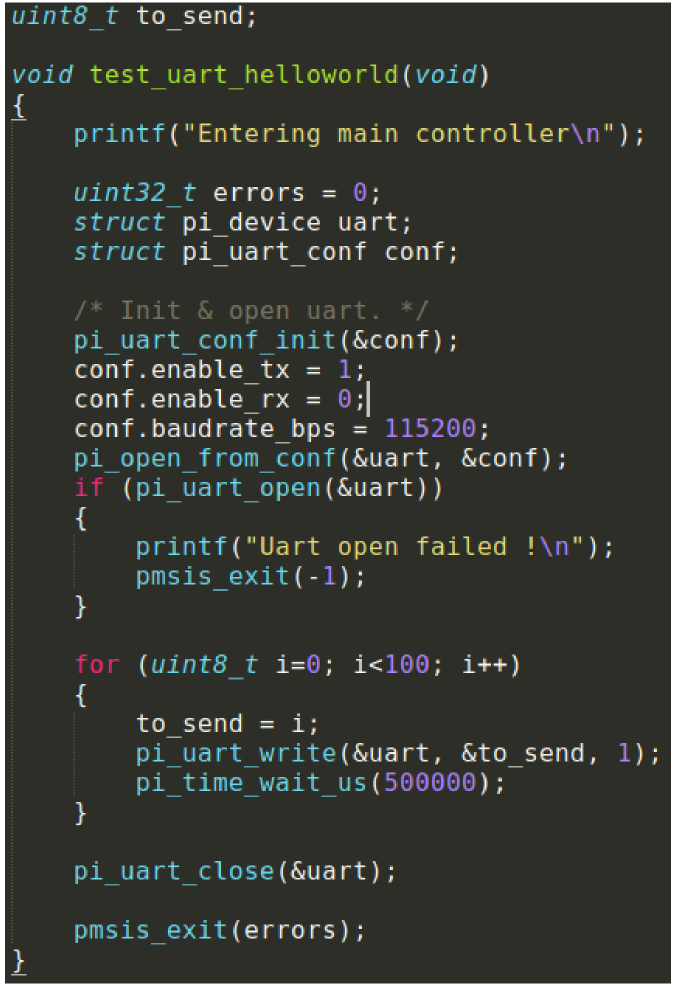
Application Example: UART and DMA



```
void USART_DMA_Start(uint32_t baudrate, uint8_t *pulpRxBuffer, uint32_t BUFFERSIZE)
{
  // Setup Communication
  USART_Config(baudrate, pulpRxBuffer, BUFFERSIZE);

  DMA_ITConfig(USARTx_RX_DMA_STREAM, DMA_IT_TC, ENABLE);

  // Enable DMA USART RX Stream
  DMA_Cmd(USARTx_RX_DMA_STREAM,ENABLE);

  // Enable USART DMA RX Requsts
  USART_DMACmd(USARTx, USART_DMAReq_Rx, ENABLE);

  // Clear DMA Transfer Complete Flags
  DMA_ClearFlag(USARTx_RX_DMA_STREAM,USARTx_RX_DMA_FLAG_TCIF);

  // Clear USART Transfer Complete Flags
  USART_ClearFlag(USARTx,USART_FLAG_TC);

  DMA_ClearFlag(USARTx_RX_DMA_STREAM, UART3_RX_DMA_ALL_FLAGS);
  NVIC_EnableIRQ(DMA1_Stream1_IRQn);
}
```

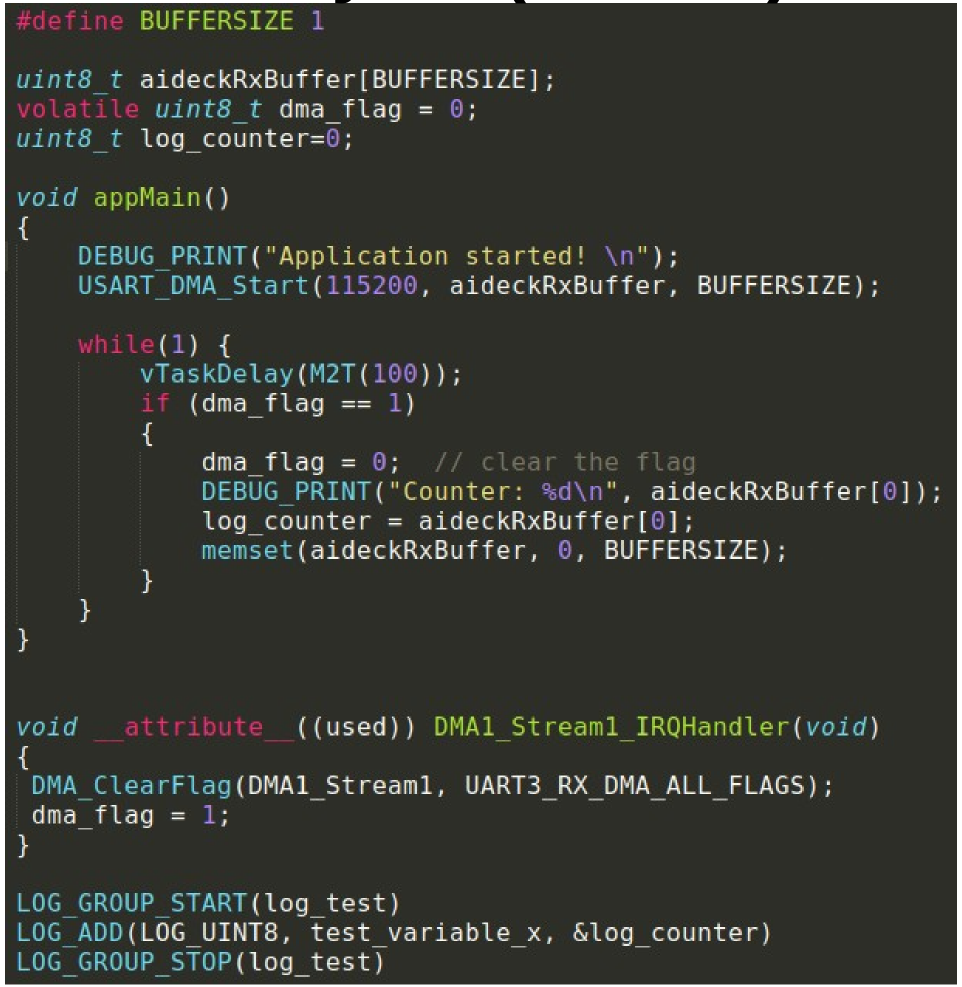
Application Example: Main

AI-Deck



```
uint8_t to_send;

void test_uart_helloworld(void)
{
    printf("Entering main controller\n");

    uint32_t errors = 0;
    struct pi_device uart;
    struct pi_uart_conf conf;

    /* Init & open uart. */
    pi_uart_conf_init(&conf);
    conf.enable_tx = 1;
    conf.enable_rx = 0;
    conf.baudrate_bps = 115200;
    pi_open_from_conf(&uart, &conf);
    if (pi_uart_open(&uart))
    {
        printf("Uart open failed !\n");
        pmsis_exit(-1);
    }

    for (uint8_t i=0; i<100; i++)
    {
        to_send = i;
        pi_uart_write(&uart, &to_send, 1);
        pi_time_wait_us(500000);
    }

    pi_uart_close(&uart);

    pmsis_exit(errors);
}
```

Crazyflie (STM32)

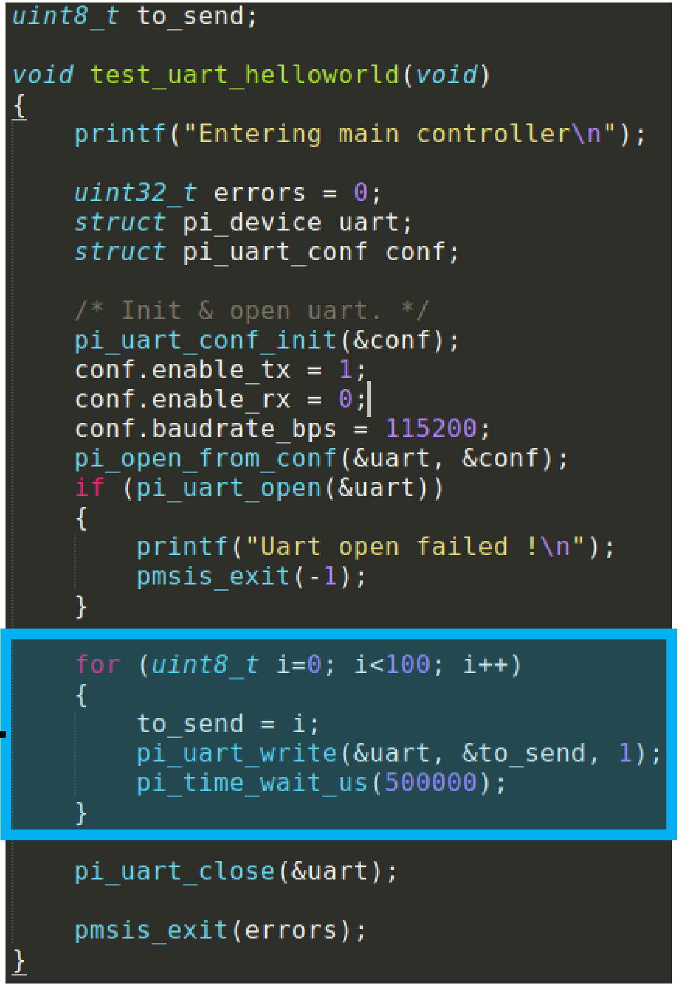


```
#define BUFFERSIZE 1

uint8_t aideckRxBuffer[BUFFERSIZE];
volatile uint8_t dma_flag = 0;
uint8_t log_counter=0;

void appMain()
{
    DEBUG_PRINT("Application started! \n");
    USART_DMA_Start(115200, aideckRxBuffer, BUFFERSIZE);

    while(1) {
        vTaskDelay(M2T(100));
        if (dma_flag == 1)
        {
            dma_flag = 0;  // clear the flag
            DEBUG_PRINT("Counter: %d\n", aideckRxBuffer[0]);
            log_counter = aideckRxBuffer[0];
            memset(aideckRxBuffer, 0, BUFFERSIZE);
        }
    }
}


void __attribute__((used)) DMA1_Stream1_IRQHandler(void)
{
 DMA_ClearFlag(DMA1_Stream1, UART3_RX_DMA_ALL_FLAGS);
 dma_flag = 1;
}

LOG_GROUP_START(log_test)
LOG_ADD(LOG_UINT8, test_variable_x, &log_counter)
LOG_GROUP_STOP(log_test)
```

Application Example: Main

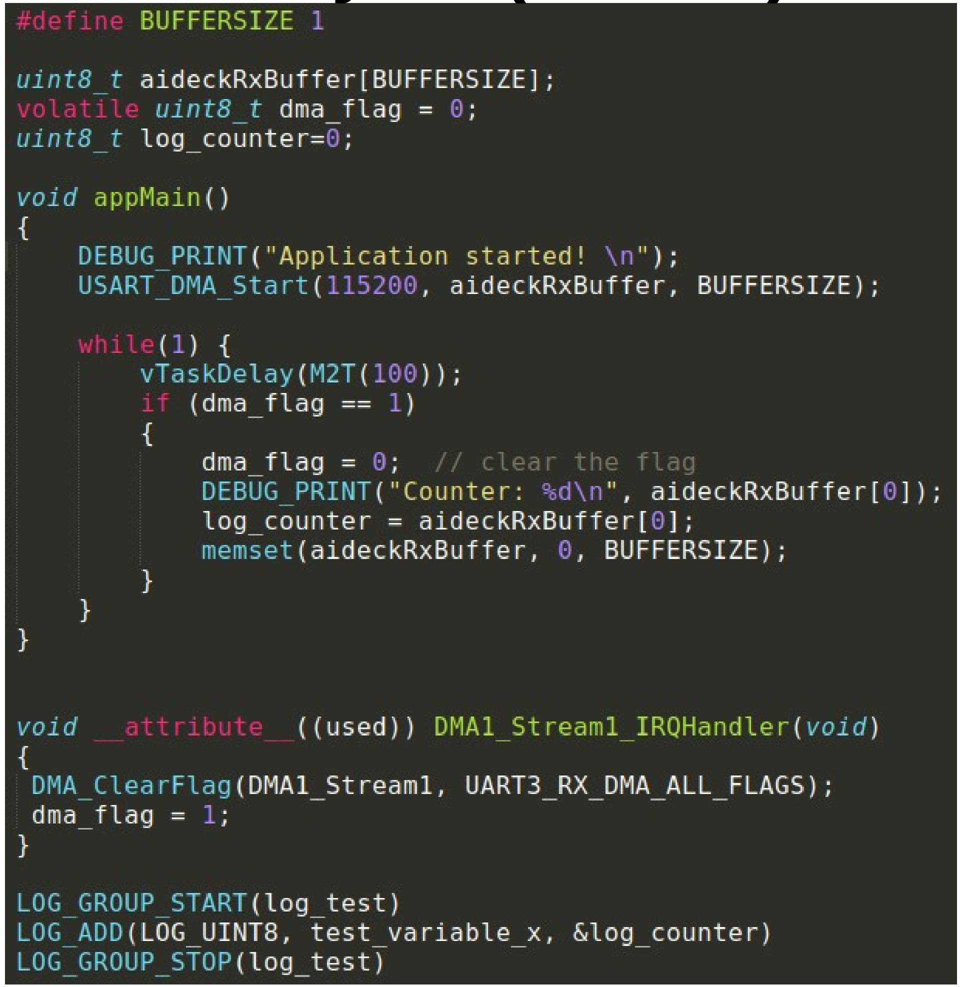
Every 0.5s: increment the counter and send its value via UART

AI-Deck



```
uint8_t to_send;

void test_uart_helloworld(void)
{
    printf("Entering main controller\n");

    uint32_t errors = 0;
    struct pi_device uart;
    struct pi_uart_conf conf;

    /* Init & open uart. */
    pi_uart_conf_init(&conf);
    conf.enable_tx = 1;
    conf.enable_rx = 0;
    conf.baudrate_bps = 115200;
    pi_open_from_conf(&uart, &conf);
    if (pi_uart_open(&uart))
    {
        printf("Uart open failed !\n");
        pmsis_exit(-1);
    }

    for (uint8_t i=0; i<100; i++)
    {
        to_send = i;
        pi_uart_write(&uart, &to_send, 1);
        pi_time_wait_us(500000);
    }

    pi_uart_close(&uart);

    pmsis_exit(errors);
}
```

Crazyflie (STM32)

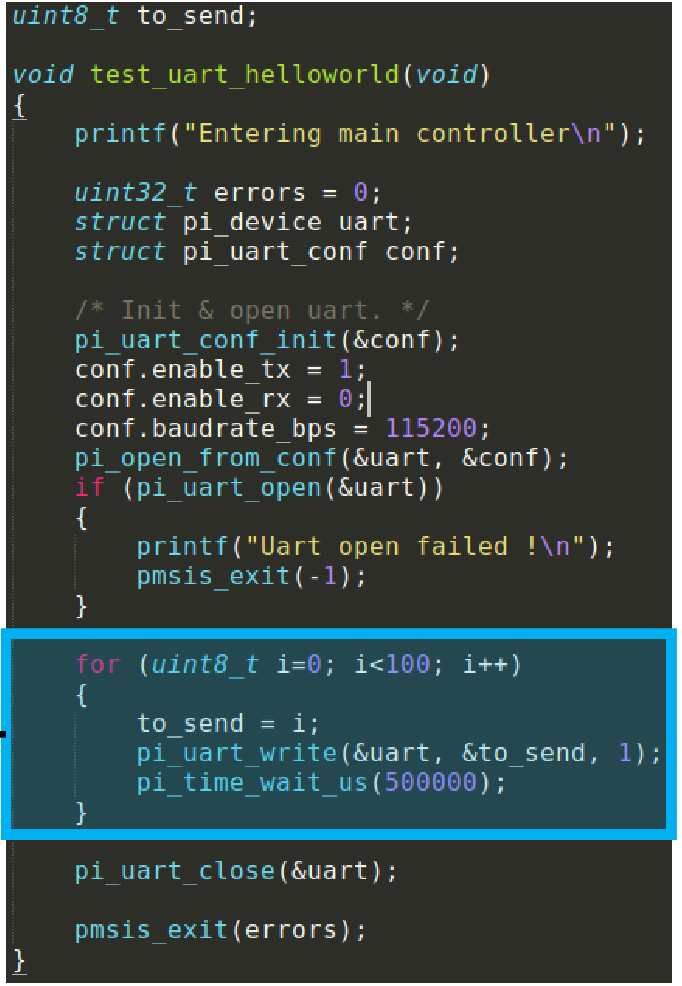


```
#define BUFFERSIZE 1

uint8_t aideckRxBuffer[BUFFERSIZE];
volatile uint8_t dma_flag = 0;
uint8_t log_counter=0;

void appMain()
{
    DEBUG_PRINT("Application started! \n");
    USART_DMA_Start(115200, aideckRxBuffer, BUFFERSIZE);

    while(1) {
        vTaskDelay(M2T(100));
        if (dma_flag == 1)
        {
            dma_flag = 0;  // clear the flag
            DEBUG_PRINT("Counter: %d\n", aideckRxBuffer[0]);
            log_counter = aideckRxBuffer[0];
            memset(aideckRxBuffer, 0, BUFFERSIZE);
        }
    }
}


void __attribute__((used)) DMA1_Stream1_IRQHandler(void)
{
 DMA_ClearFlag(DMA1_Stream1, UART3_RX_DMA_ALL_FLAGS);
 dma_flag = 1;
}

LOG_GROUP_START(log_test)
LOG_ADD(LOG_UINT8, test_variable_x, &log_counter)
LOG_GROUP_STOP(log_test)
```

Application Example: Main

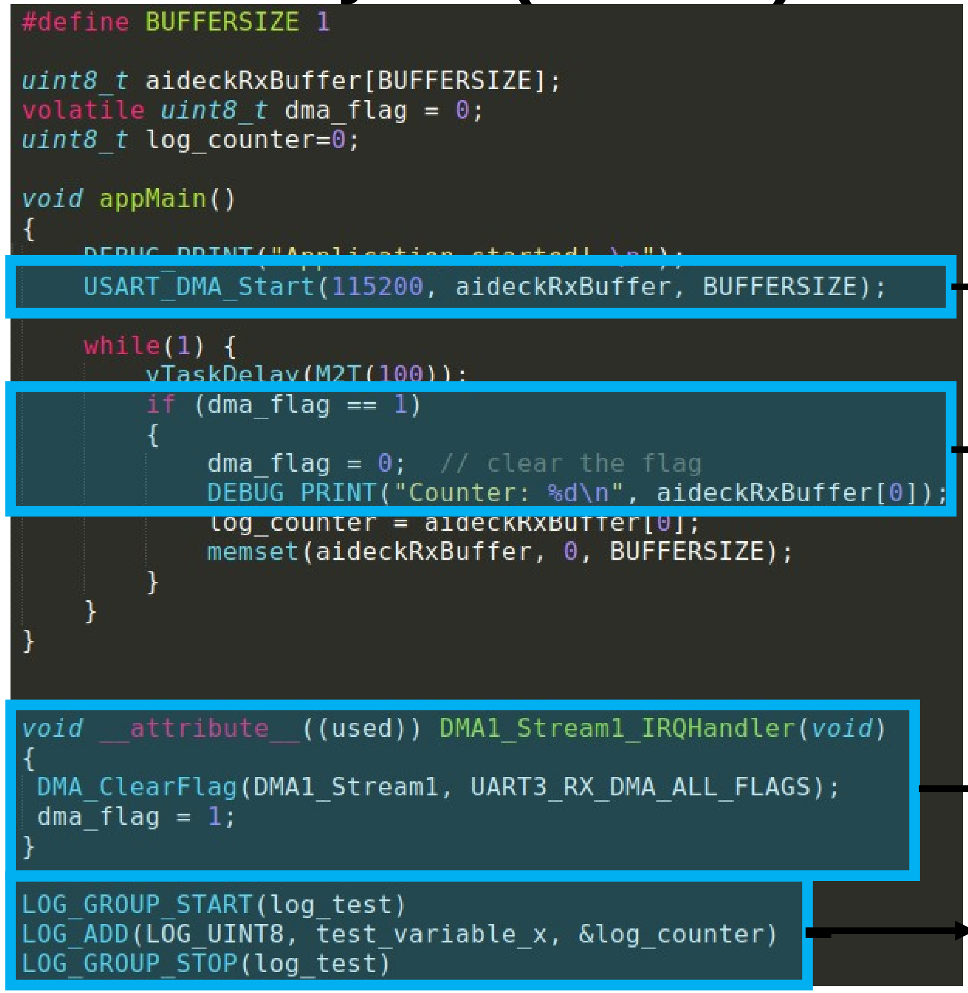
AI-Deck

Every 0.5s: increment the counter and send its value via UART



```c
uint8_t to_send;

void test_uart_helloworld(void)
{
    printf("Entering main controller\n");

    uint32_t errors = 0;
    struct pi_device uart;
    struct pi_uart_conf conf;

    /* Init & open uart. */
    pi_uart_conf_init(&conf);
    conf.enable_tx = 1;
    conf.enable_rx = 0;
    conf.baudrate_bps = 115200;
    pi_open_from_conf(&uart, &conf);
    if (pi_uart_open(&uart))
    {
        printf("Uart open failed !\n");
        pmsis_exit(-1);
    }

    for (uint8_t i=0; i<100; i++)
    {
        to_send = i;
        pi_uart_write(&uart, &to_send, 1);
        pi_time_wait_us(500000);
    }

    pi_uart_close(&uart);

    pmsis_exit(errors);
}
```

Crazyflie (STM32)



```c
#define BUFFERSIZE 1

uint8_t aideckRxBuffer[BUFFERSIZE];
volatile uint8_t dma_flag = 0;
uint8_t log_counter=0;

void appMain()
{
    DEBUG_PRINT("Application started! \n");
    USART_DMA_Start(115200, aideckRxBuffer, BUFFERSIZE);

    while(1) {
        vTaskDelay(M2T(100));
        if (dma_flag == 1)
        {
            dma_flag = 0;  // clear the flag
            DEBUG_PRINT("Counter: %d\n", aideckRxBuffer[0]);
            log_counter = aideckRxBuffer[0];
            memset(aideckRxBuffer, 0, BUFFERSIZE);
        }
    }
}

void __attribute__((used)) DMA1_Stream1_IRQHandler(void)
{
 DMA_ClearFlag(DMA1_Stream1, UART3_RX_DMA_ALL_FLAGS);
 dma_flag = 1;
}

LOG_GROUP_START(log_test)
LOG_ADD(LOG_UINT8, test_variable_x, &log_counter)
LOG_GROUP_STOP(log_test)
```

Init DMA and UART

If the flag is set, print the received value

DMA “full buffer” interrupt

Define log

Hands-on

Hands-on demonstration of the system's functionality