STM32WL & Lorawan

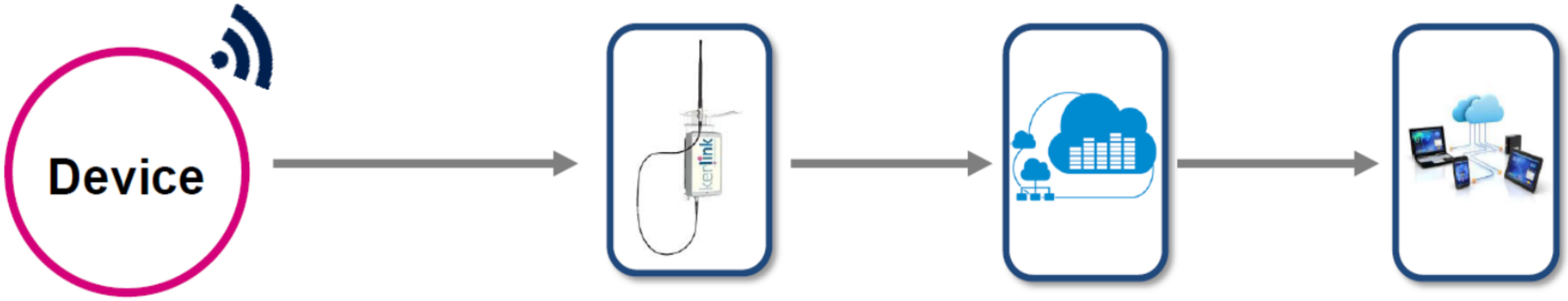
Hands-on: sending data to LoRaWAN network

Hands-on

Key learning

- LoRaWAN ® protocol stack implementation details
- Sequencer & Low Power Mode
- LoRaWAN ® application flow
- Uplink & downlink data transmission
- Project source code generation using STM32CubeIDE (STM32CubeMX)

What does LoRaWAN network consists of ?



End Point

Gateway

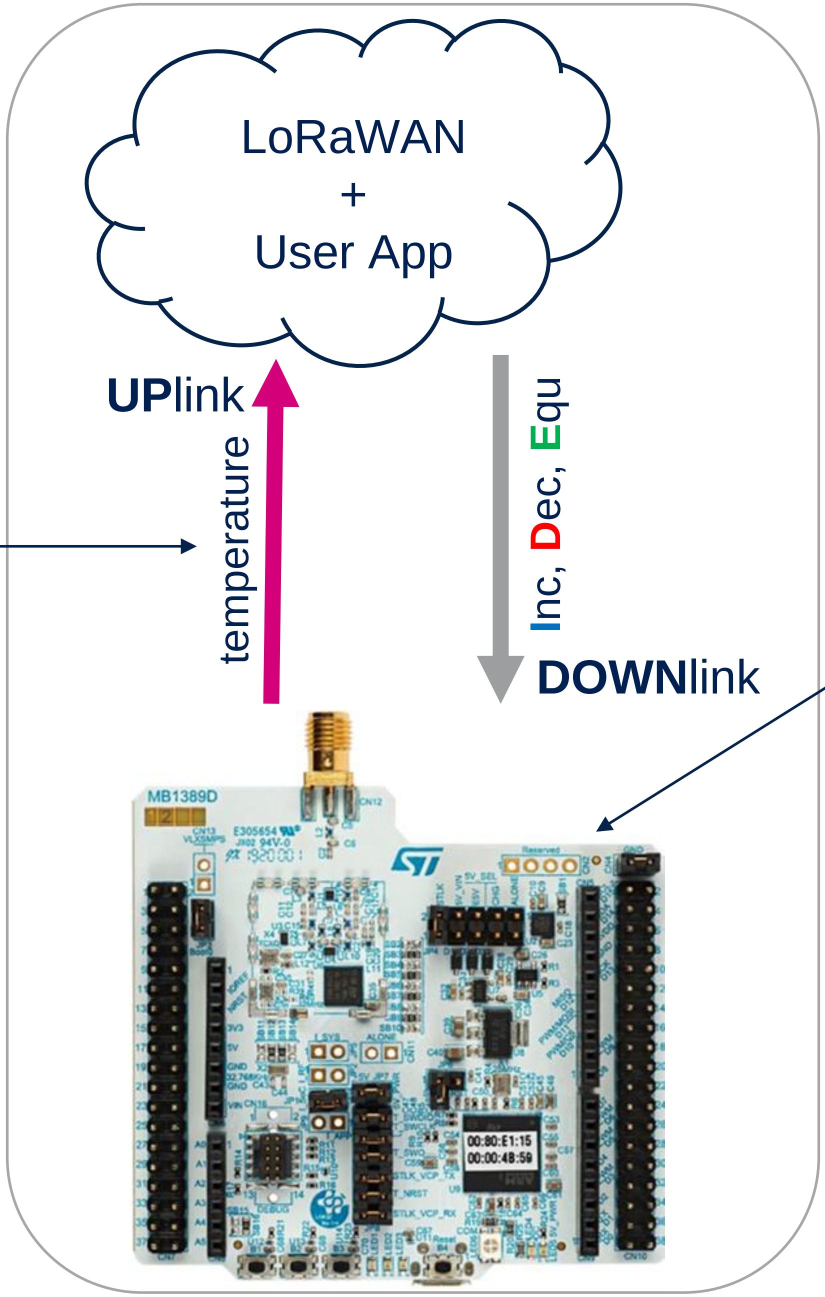
Network server

Application server

LoRaWAN temperature sensor

Hands-on

STM32WL internal temperature sensor connected to ADC mux

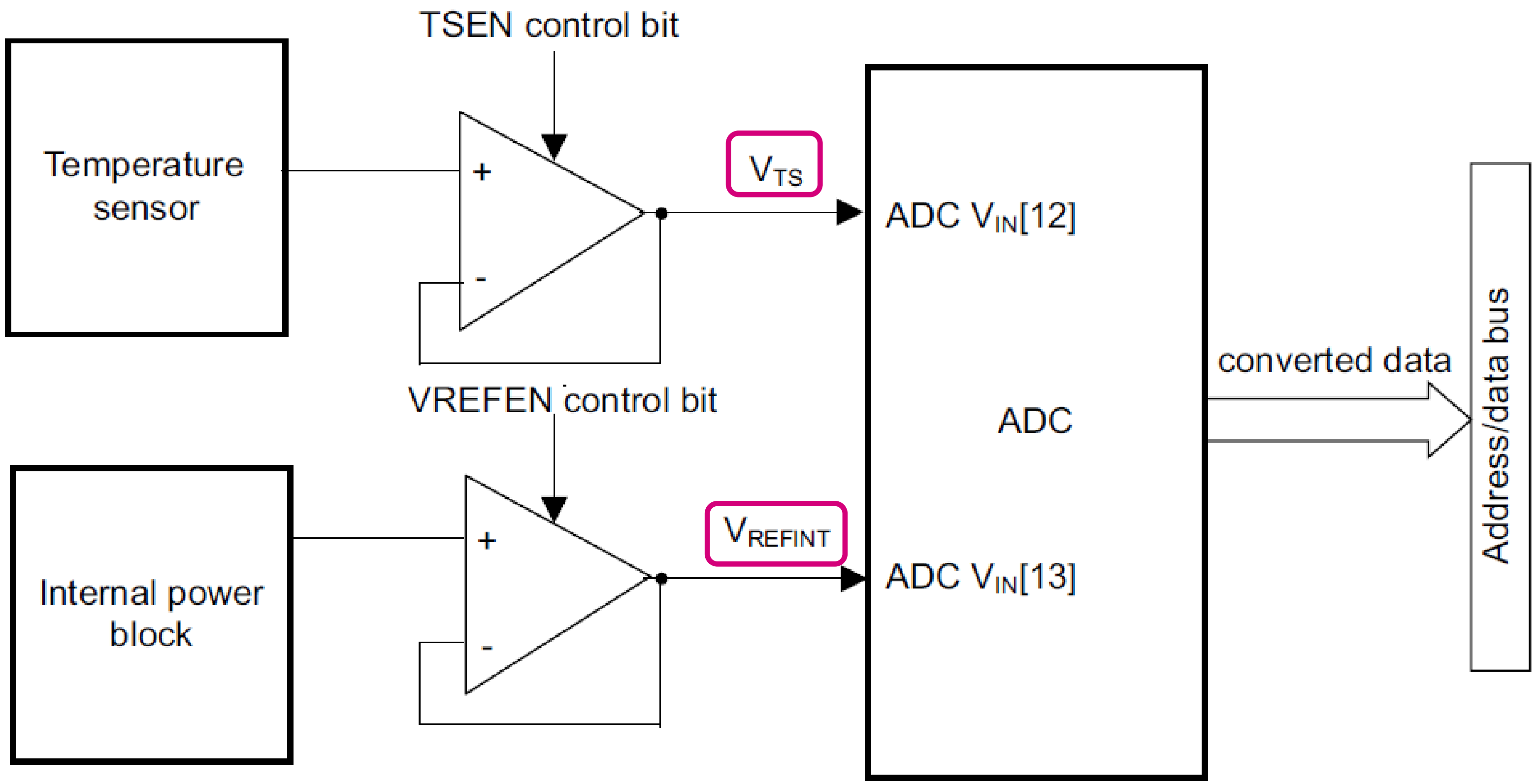


STM32WL Nucleo-64

- SMA antenna connector
- Arduino™ extension connectors : easy access to add-ons
- STM32WL (under a mettalic shield)
- Integrated ST-LINK/V3: mass storage device flash programming
- 4 push buttons including reset, 3 LEDs, jumper settings
- Flexible board power supply: through USB or external source

life.augmented

STM32WL5 internal temperature sensor




```
/* #warning "to be replaced by __LL_ADC_CALC_TEMPERATURE when calibration data will be set in prod"*/
  temperatureDegreeC = __LL_ADC_CALC_TEMPERATURE_TYP_PARAMS(TEMPSENSOR_TYP_AVGSLOPE,
                                                             TEMPSENSOR_TYP_CAL1_V,
                                                             TEMPSENSOR_CAL1_TEMP,
                                                             batteryLevelmV,
                                                             measuredLevel,
                                                             LL_ADC_RESOLUTION_12B);
```

Hands-on: protocol stack provider

Semtech provides LoRaWAN protocol stack sw

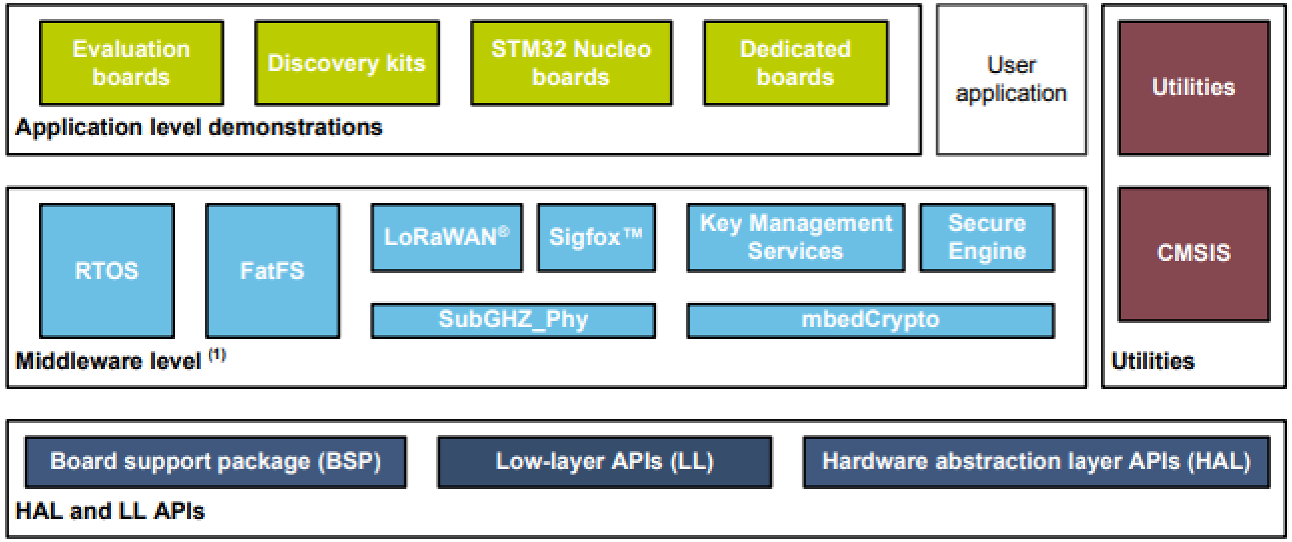
The LoRaWAN stack for STM32WL is LoRaWAN L2 V1.0.3 compatible

- Based on **LoRaMac-node** from Semtech on github
- Supported features
    - Class A, Class C (Unicast and Multicast),
    - Class B (Unicast / Multicast)

life.augmented

STM32CubeWL v1.0.0

The package includes low-layer (LL) and hardware abstraction layer (HAL) APIs that cover the microcontroller hardware, together with an extensive set of middleware and examples running on STMicroelectronics boards.



See for details:
UM2643 “Getting started with STM32CubeWL for STM32WL Series”
AN5409 “STM32Cube MCU Package examples for STM32WL Series”

life.augmented

STM32CubeWL v1.0.0

Let's focus on LoRa(WAN) examples

I-CUBE-LRWAN migrated examples:

- Ping-Pong application (Point-Point Phy)
- **End-Node** application
- AT-Slave application (AT-Commands modem)

Middleware

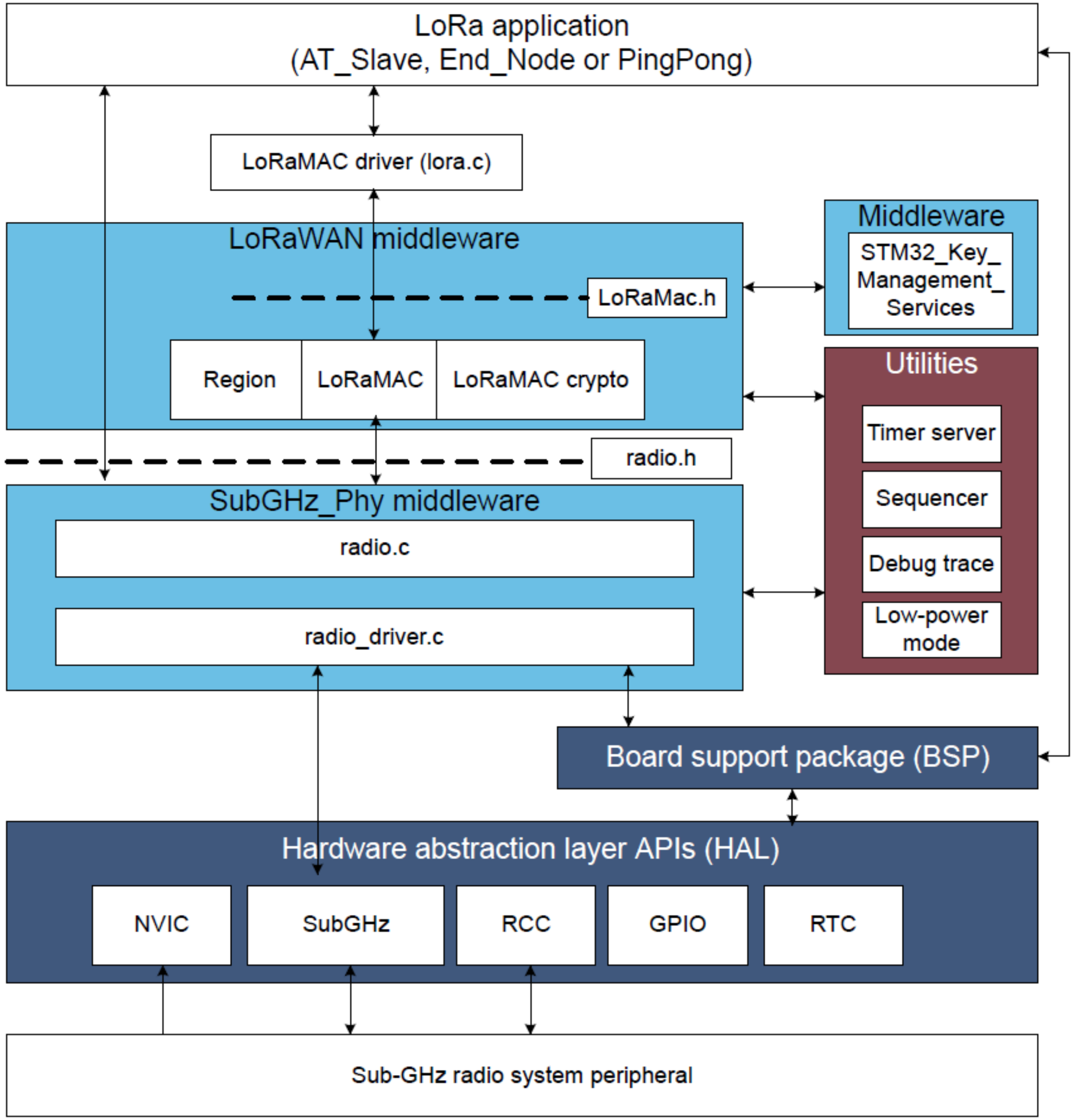
- LoRaWAN contains the Mac layer
- SubGHz_Phy contains the Phy Layer

Utilities

- Generic Utilities will be common with STM32WB

BSP

- Drives the RF switch, board configuration



life.augmented

See for details:
UM2643 Getting started with STM32CubeWL for STM32WL Series

STM32Cube_FW_WL_V1.0.0 End_Node example details

App: application source files,
to be edited during hands-on: lora_app.c

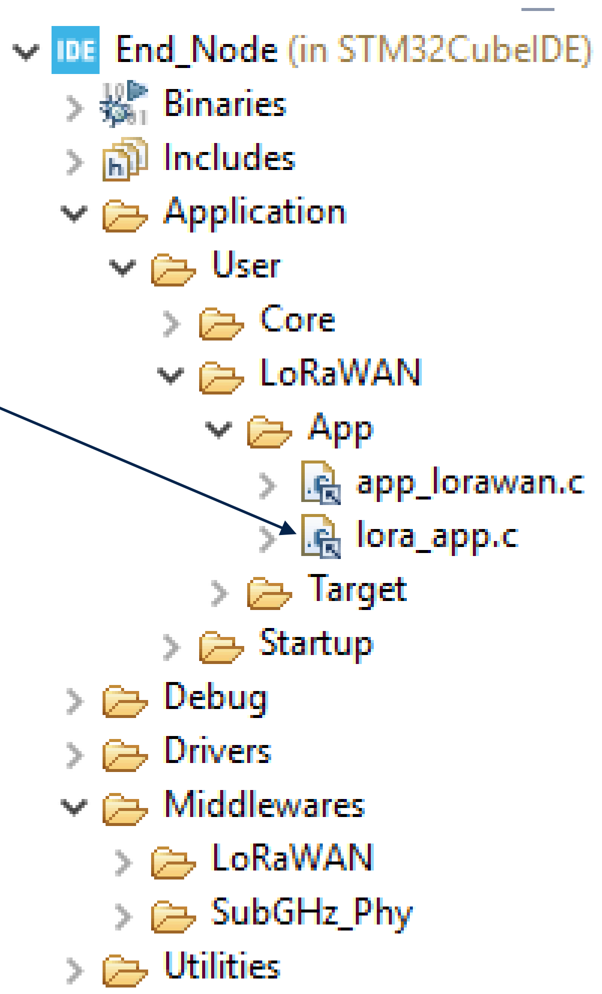
Core: MCU specific source files

Drivers: BSP, CMSIS & HAL drivers

Middlewares: LoRa MAC, LoRa FSM (regular & test mode), crypto & utilities

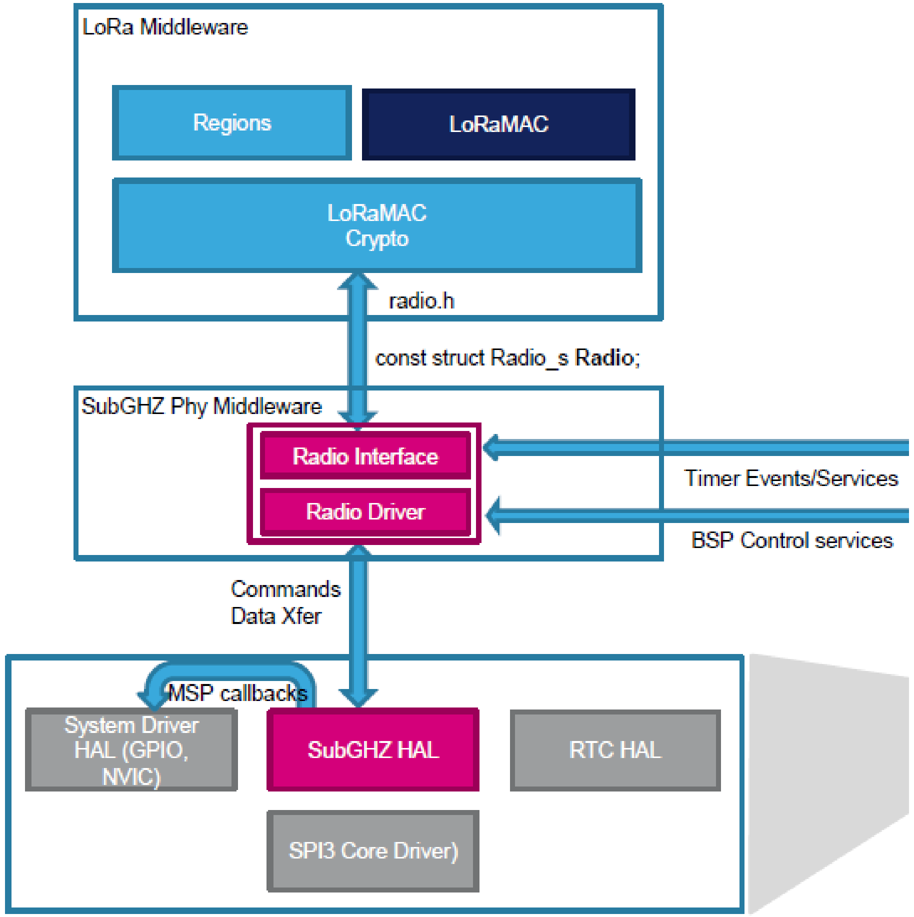
SubGHz_Phy: SubGHz phy radio middleware

Utilities: LPM manager, sequencer, sw timers, trace, system time, …



SubGHz_Phy middleware interaction model

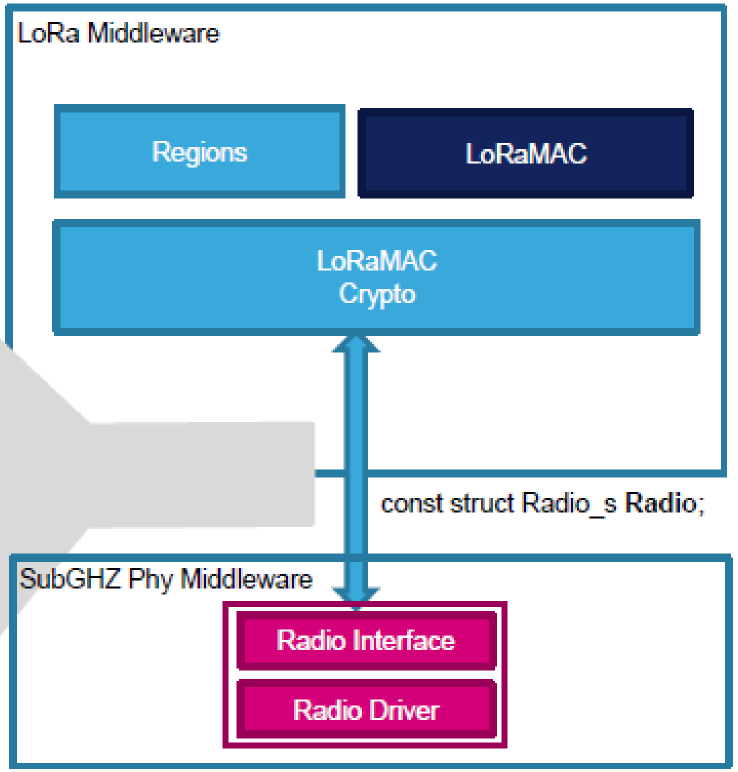
The diagram shows the interaction model of the LPWAN middleware (LoRaWAN Stack) with common SubGHz_Phy layer and the low-level drivers



SubGHz_Phy middleware interaction model

- The LoRaWan Middleware interacts with the Radio RF through the Radio_s structure fops

```
const struct Radio_s Radio =
{
    RadioInit,
    RadioGetStatus,
    RadioSetModem,
    RadioSetChannel,
    RadioIsChannelFree,
    RadioRandom,
    RadioSetRxConfig,
    RadioSetTxConfig,
    RadioBoardCheckRfFrequency,
    RadioGetTimeOnAir,
    RadioSend,
    RadioSleep,
    RadioStandby,
    RadioRx,
    RadioStartCad,
    RadioSetTxContinuousWave,
    RadioRssi,
    RadioWrite,
    RadioRead,
    RadioSetMaxPayloadLength,
    RadioSetPublicNetwork,
    RadioGetWakeUpTime
};
```

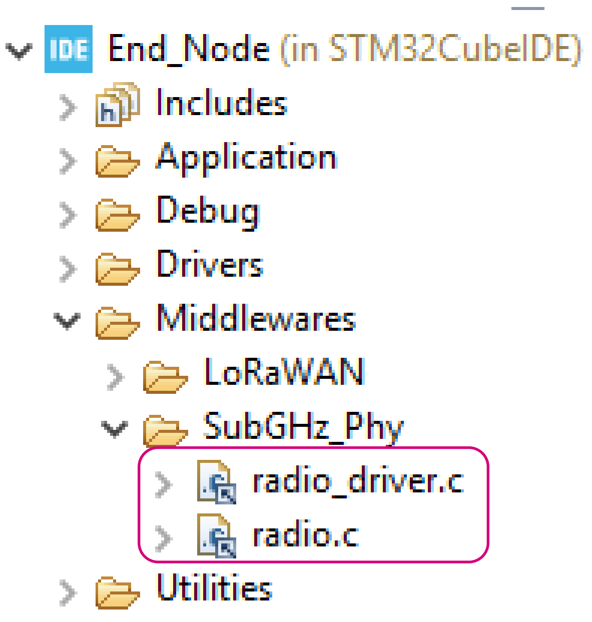


life.augmented

Middleware radio interface

- **radio.h**
  - interface for the LoRaMAC layer or Application
  - Compatible with source file radio.h of I-CUBE-LRWAN

- Radio is split into 2 levels
  - Radio high level
    - Source file: **radio.c**
    - SubGHz_phy middleware interface with higher level sw
  - Radio low level functions
    - Source file: **radio_driver.c**
    - Interface with
      - stm32wlxx_hal_subg.h for SubGHz services
      - stm32wlxx_nucleo.h for RF BSP services



Files: radio.c and radio_driver.c remains comparable with legacy. (Easy to update changes form Semtech)

Example for **TX** using radio interface

```
// Radio initialization
RadioEvents.TxDone = OnTxDone;
RadioEvents.RxDone = OnRxDone;
RadioEvents.TxTimeout = OnTxTimeout;
RadioEvents.RxTimeout = OnRxTimeout;
RadioEvents.RxError = OnRxError;
Radio.Init( &RadioEvents );

// Radio Tx Configuation in Lora mode
Radio.SetTxConfig( MODEM_LORA, TX_OUTPUT_POWER, 0, LORA_BANDWIDTH,
                   LORA_SPREADING_FACTOR, LORA_CODINGRATE,
                   LORA_PREAMBLE_LENGTH, LORA_FIX_LENGTH_PAYLOAD_ON,
                   true, 0, 0, LORA_IQ_INVERSION_ON, 3000 );

// Radio Set Rf frequency
Radio.SetChannel( RF_FREQUENCY );

// Radio send a buffer
Radio.Send( Buffer, BufferSize );
```

Example for **RX** using radio interface

```
 // Radio initialization
 RadioEvents.TxDone = OnTxDone;
 RadioEvents.RxDone = OnRxDone;
 RadioEvents.TxTimeout = OnTxTimeout;
 RadioEvents.RxTimeout = OnRxTimeout;
 RadioEvents.RxError = OnRxError;
 Radio.Init( &RadioEvents );

 // Radio Rx Configuation in FSK mode
 Radio.SetRxConfig( MODEM_FSK, FSK_BANDWIDTH, FSK_DATARATE,
                    0, FSK_AFC_BANDWIDTH, FSK_PREAMBLE_LENGTH,
                    0, FSK_FIX_LENGTH_PAYLOAD_ON, 0, true,
                    0, 0,false, true );

// Radio Set Rf frequency
Radio.SetChannel( RF_FREQUENCY );

// Radio set in Rx mode
Radio.Rx( RX_TIMEOUT_VALUE );
```

Radio low level functions: stm32wlxx_hal_subg

```
void HAL_SUBGHZ_ExecSetCmd(SUBGHZ_RadioSetCmd_t command, uint8_t *buffer, uint16_t size);
```

Example:

byte 0	bits 7:0	**Opcode**: 0x84
byte 1	bits 7:3	Reserved, must be kept at reset value.
	bit 2	**SleepCfg_Start**: Sub-GHz radio startup selection 0: cold startup when exiting Sleep mode, configuration registers reset 1: warm startup when exiting Sleep mode, configuration registers kept in retention Note: *Only the configuration of the activated modem, before going to Sleep mode, is retained. The configuration of the other modes is lost and must be re-configured when exiting Sleep mode.*
	bit 1	Reserved, must be kept at reset value.
	bit 0	**SleepCfg_RTCEn**: Sub-GHz radio RTC wakeup enable 0: Sub-GHz radio RTC wakeup disabled 1: Sub-GHz radio RTC wakeup enabled

```
void SUBGRF_SetSleep( SleepParams_t sleepConfig )
{
    BSP_RF_SW_Reset(); /* switch the antenna OFF by SW */
    HAL_SUBGHZ_ExecSetCmd( RADIO_SET_SLEEP, &sleepConfig.Value, 1 );
    OperatingMode = MODE_SLEEP;
}
```

```
typedef enum SUBGHZ_RadioSetCmd_e
{
  RADIO_SET_SLEEP                          = 0x84,
  RADIO_SET_STANDBY                        = 0x80,
  RADIO_SET_FS                             = 0xC1,
  RADIO_SET_TX                             = 0x83,
  RADIO_SET_RX                             = 0x82,
  RADIO_SET_RXDUTYCYCLE                    = 0x94,
  RADIO_SET_CAD                            = 0xC5,
  RADIO_SET_TXCONTINUOUSWAVE               = 0xD1,
  RADIO_SET_TXCONTINUOUSPREAMBLE           = 0xD2,
  RADIO_SET_PACKETTYPE                     = 0x8A,
  RADIO_SET_RFFREQUENCY                    = 0x86,
  RADIO_SET_TXPARAMS                       = 0x8E,
  RADIO_SET_PACONFIG                       = 0x95,
  RADIO_SET_CADPARAMS                      = 0x88,
  RADIO_SET_BUFFERBASEADDRESS              = 0x8F,
  RADIO_SET_MODULATIONPARAMS               = 0x8B,
  RADIO_SET_PACKETPARAMS                   = 0x8C,
  RADIO_RESET_STATS                        = 0x00,
  RADIO_CFG_DIOIRQ                         = 0x08,
  RADIO_CLR_IRQSTATUS                      = 0x02,
  RADIO_CALIBRATE                          = 0x89,
  RADIO_CALIBRATEIMAGE                     = 0x98,
  RADIO_SET_REGULATORMODE                  = 0x96,
  RADIO_SET_TCXOMODE                       = 0x97,
  RADIO_SET_TXFALLBACKMODE                 = 0x93,
  RADIO_SET_RFSWITCHMODE                   = 0x9D,
  RADIO_SET_STOPRXTIMERONPREAMBLE          = 0x9F,
  RADIO_SET_LORASYMBTIMEOUT                = 0xA0,
} SUBGHZ_RadioSetCmd_t;
```

Radio low level functions: stm32wlxx_hal_subg

```
void HAL_SUBGHZ_ExecGetCmd(SUBGHZ_RadioGetCmd_t command, uint8_t *buffer, uint16_t size);
```

Example:

Command	Opcode	Parameters
Get_Status()	0xC0	Status

```
RadioStatus_t SUBGRF_GetStatus( void )
{
    uint8_t stat = 0;
    RadioStatus_t status;
    HAL_SUBGHZ_ExecGetCmd( RADIO_GET_STATUS, ( uint8_t * )&stat, 1 );
    status.Value = stat;
    return status;
}
```

Get_RssiInst()	0x15	Status, RssiInst

```
int8_t SUBGRF_GetRssiInst( void )
{
    uint8_t buf[1];
    int8_t rssi = 0;

    HAL_SUBGHZ_ExecGetCmd( RADIO_GET_RSSIINST, buf, 1 );
    rssi = -buf[0] >> 1;
    return rssi;
}
```

```
typedef enum SUBGHZ_RadioGetCmd_e
{
  RADIO_GET_STATUS              = 0xC0,
  RADIO_GET_PACKETTYPE          = 0x11,
  RADIO_GET_RXBUFFERSTATUS      = 0x13,
  RADIO_GET_PACKETSTATUS        = 0x14,
  RADIO_GET_RSSIINST            = 0x15,
  RADIO_GET_STATS               = 0x10,
  RADIO_GET_IRQSTATUS           = 0x12,
  RADIO_GET_ERROR               = 0x17,
} SUBGHZ_RadioGetCmd_t;
```

life.augmented

Radio basic registers

Registers can be accessed using

- `HAL_SUBGHZ_WriteRegisters(uint16_t address, uint8_t *buffer, uint16_t size);`
- `HAL_SUBGHZ_ReadRegisters(uint16_t address, uint8_t *buffer, uint16_t size);`

Radio Interface configures these registers.

Name	length	description
SUBGHZ_GBSYNCR **(REG_BIT_SYNC)**	1	This register must be cleared to 0x00 when using packet types other than LoRa.
SUBGHZ_GPKTCTL1AR **(REG_LR_WHITSEEDBASEADDR_MSB)**	1	Set continuous packet generation mode Bit 5 **SYNCDETEN:** Generic packet synchronization word detection enable Bit 4 **CONTTX:** Generic packet continuous transmit enable Bits 3:2 **INFSEQSEL[1:0]:** Generic packet infinite sequence selection 0x5555 or all zero 0x0000 or all 1 or PRBS9 Bit 1 **INFSQEQEN:** Generic packet infinite sequence enable Bit 0 **WHITEINI[8]:** Generic packet whitening initial value MSB bit [8] 7
SUBGHZ_GPKTCTL1AR **(REG_LR_WHITSEEDBASEADDR_LSB)**	1	Set Whitening Seed **WHITEINI[7:0]** for GFSK packet
SUBGHZ_GCRCINI **(REG_LR_CRCSEEDBASEADDR)**	2	Set Crc Seed for GFSK packet
SUBGHZ_GCRCPOLRH **(REG_LR_CRCPOLYBASEADDR)**	2	Set Crc Polynomial for GFSK packet
SUBGHZ_GSYNCR **(REG_LR_SYNCWORDBASEADDRESS)**	8	Set the Sync Word for GFSK
SUBGHZ_LSYNCRL **(REG_LR_SYNCWORD)**	2	LoRa synchronization word (public or private)
SUBGHZ_RNGR **(RANDOM_NUMBER_GENERATORBASEADDR)**	4	Random generator read value
RXGAINCR **(REG_RX_GAIN)**	1	receiver gain control register. Use for normal or boosted gain
SUBGHZ_HSEINTRIMR **REG_XTA_TRIM**	1	Xtal internal Cap trimming value (xtb also exist)
SUBGHZ_PAOCPR **REG_OCP**	1	Set maximum current for Over Current Protection.

life.augmented

Radio board services (BSP for custom board)

Hands-on

NUCLEO-STM32WL55JC1 RF API (radio_board_if.c/.h)

Init / Deinit Radio Switch (RX/TX, LP/HP)
`RBI_Init() /  RBI_DeInit()`

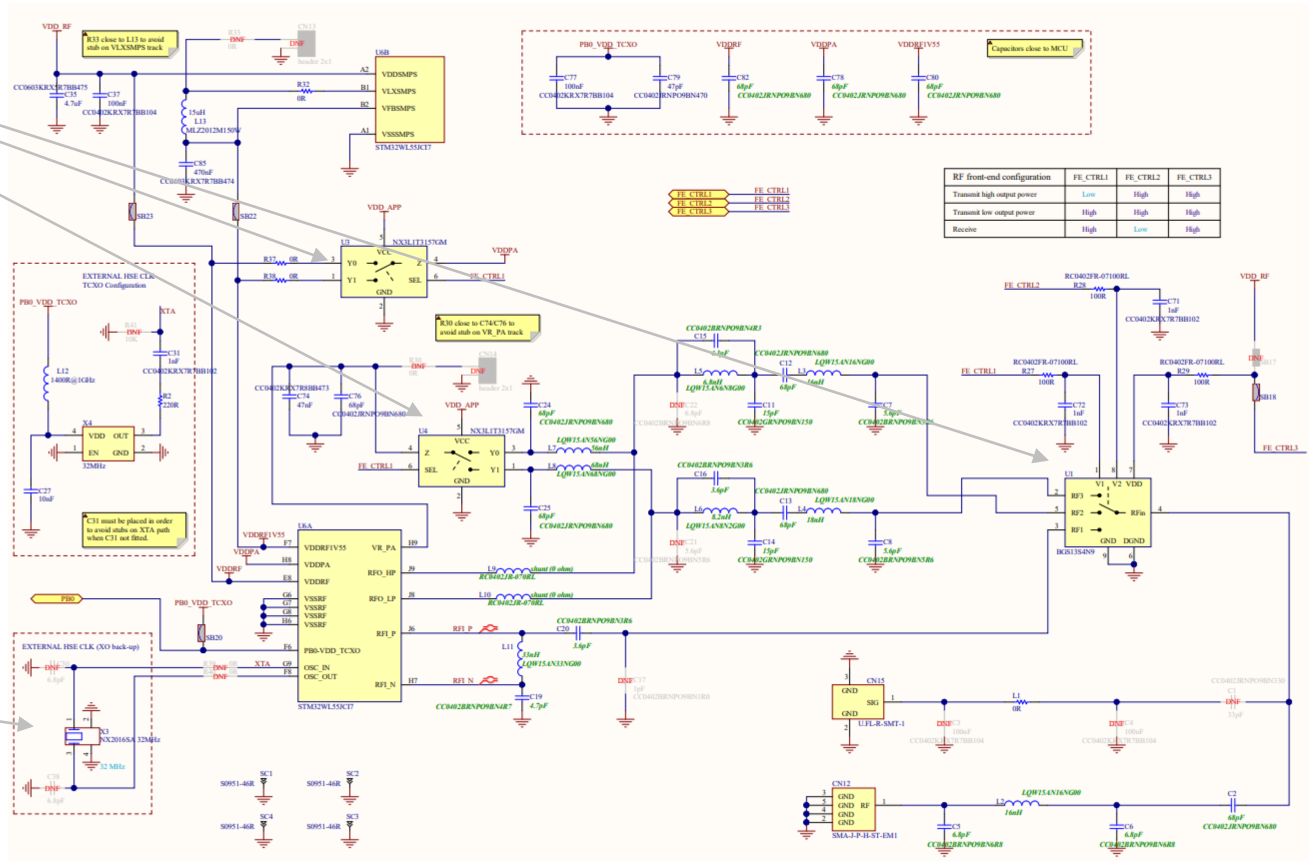
Configure Radio Switch (OFF, RX/TX, LP/HP)
`RBI_ConfigRFSwitch()`

Get TX config (LP/HP or LP only or HP only)
`RBI_GetTxConfig()`

Get Radio wakeup time [ms]
`RBI_GetWakeUpTime()`

Check if TCXO is supported
`RBI_IsTCXO()`

Check if TCXO is supported
`RBI_IsDCDC()`



life.augmented

Utilities

Hands-on

Trace: DMA trace

Uses circular buffer and DMA to print in real time, basic tool for app activity logging

Lpm :low power manager

Centralizes low power requirements from module, and go in appropriate low power

E.g. when Trace printing, MCU shall not go in STOP mode

Sequencer: previously known has scheduler

Framework to safely go in low power

Records manages tasks and events

Timer: this is the timer list or server

Used by middleware and application

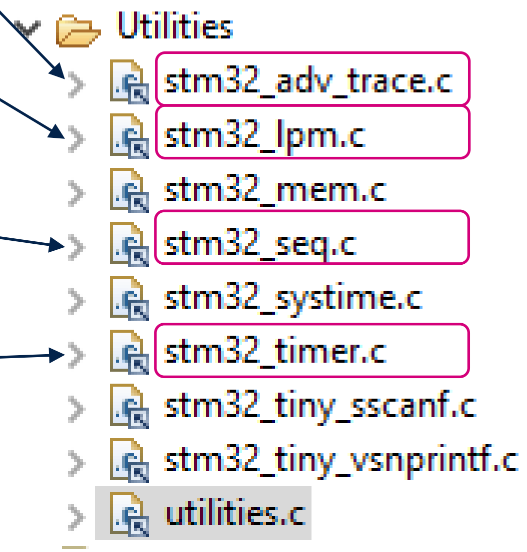
stm32_mem.c : memory operations (copy,compare, set)

stm32_systime : set / get the system time

stm32_tiny_sscanf.c : low footprint scanf

stm32_tiny_vsnprintf.c : low footprint printf

utilities.c : rnd, memory operations, hex conversion

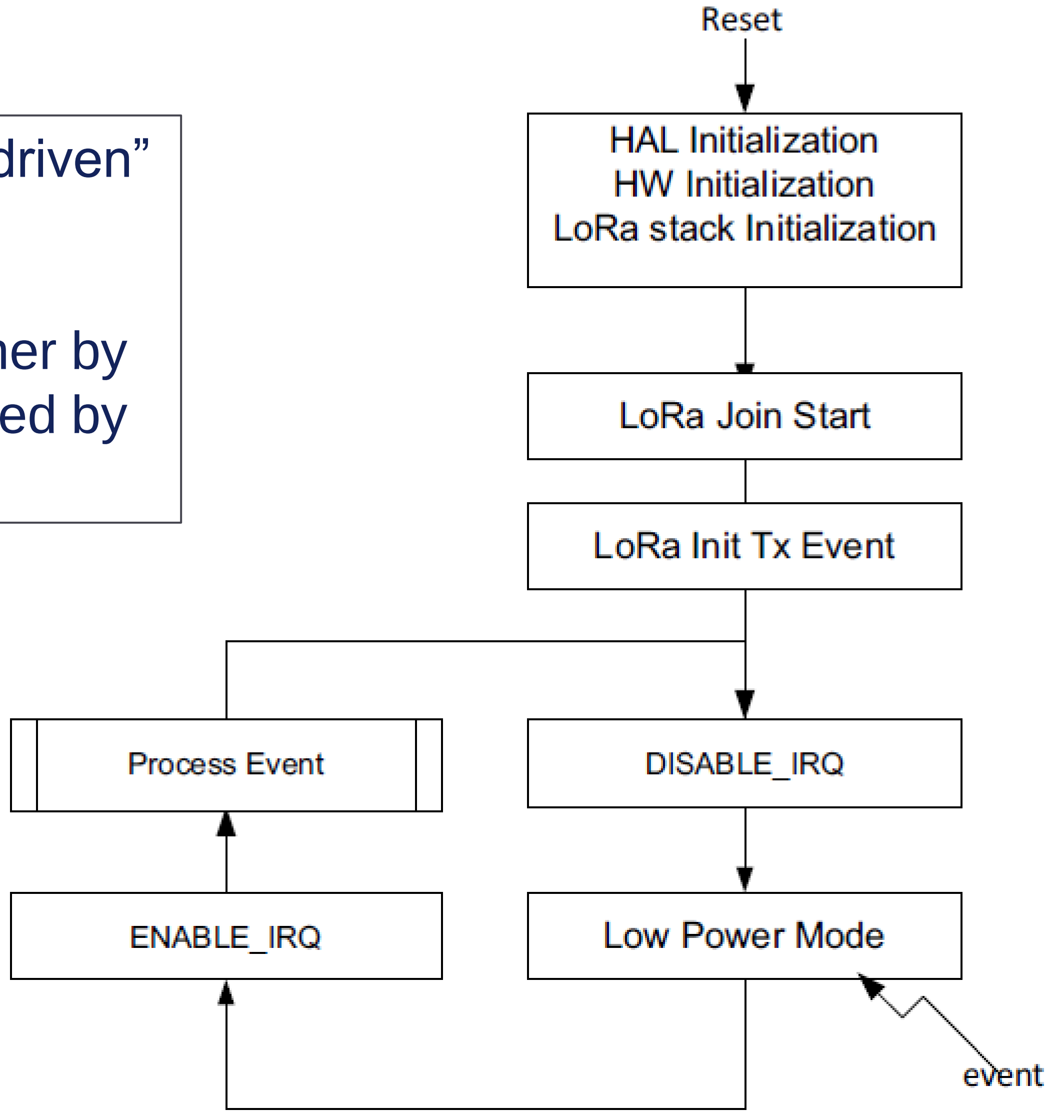


We have the bricks, let’s implement the application...

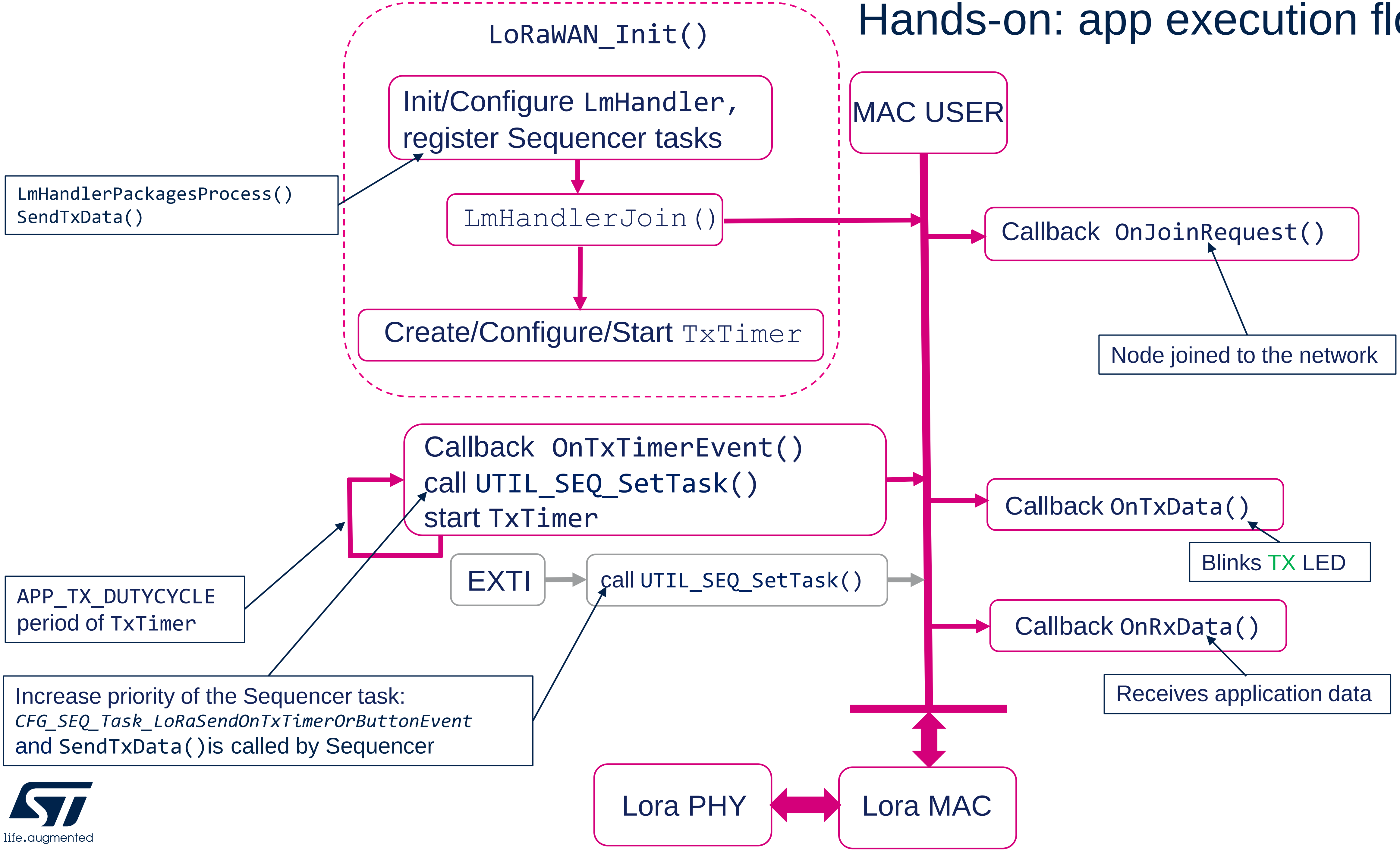
Hands-on: app execution flow

The operational model is based on “event-driven” paradigms including “time-driven”.

The behavior of the system is triggered either by a timer event or by radio event and controlled by Sequencer.



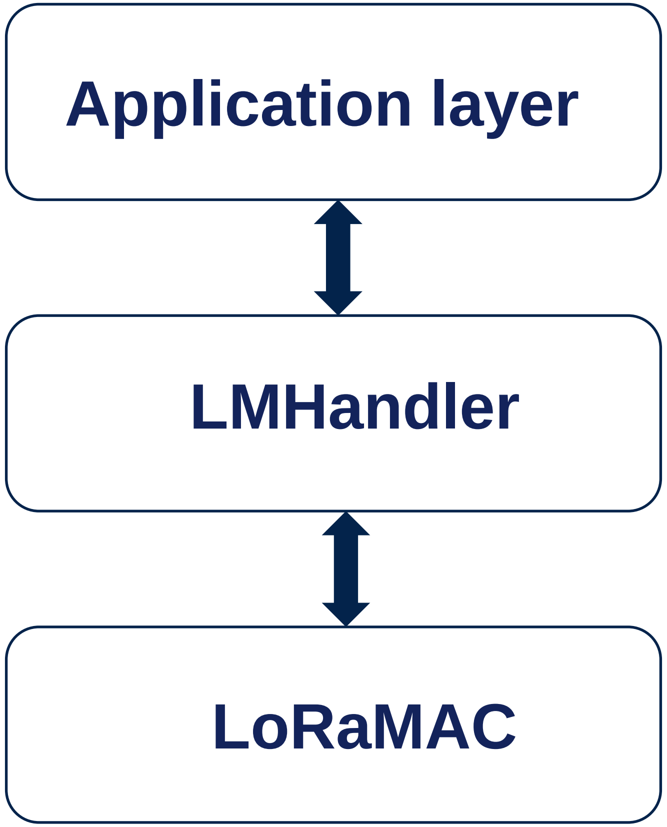
Hands-on: app execution flow



life.augmented

Hands-on: LmHandler

LmHandler API: LoRaMAC layer handling in **LmHandler.c**



Examples of LmHandler API:

- **`LmHandlerSend();`** TX data to the network,
- **`LmHandlerJoin();`** request to join the network,
- **`LmHandlerRequestClass();`** change the node class,
- **`LmHandlerSetAdrEnable();`** set Adaptive Data Rate,
- **`LmHandlerSetDevEUI();`** set Device Unique ID,
- ...

Hands-on: callbacks

Application callbacks structure variable definition in **lora_app.c**

```
static LmHandlerCallbacks_t LmHandlerCallbacks =
{
  .GetBatteryLevel =              GetBatteryLevel,
  .GetTemperature =               GetTemperatureLevel,
  .OnMacProcess =                 OnMacProcessNotify,
  .OnJoinRequest =                OnJoinRequest,
  .OnTxData =                     OnTxData,
  .OnRxData =                     OnRxData
};
```

Hands-on: Callbacks

Example: on MAC layer event the relevant callback increases the priority of the Finite State Machine of Lorawan process Sequencer task what triggers the task execution.

```
static void OnTxTimerEvent(void *context)
{
  UTIL_SEQ_SetTask((1 << CFG_SEQ_Task_LoRaSendOnTxTimerOrButtonEvent), CFG_SEQ_Prio_0);
  UTIL_TIMER_Start(&TxTimer);
}
```

life.augmented

Hands-on: Sequencer

Sequencer tasks must be registered during app initialization phase

```
UTIL_SEQ_RegTask((1 << CFG_SEQ_Task_LmHandlerProcess), UTIL_SEQ_RFU, LmHandlerProcess);
UTIL_SEQ_RegTask((1 << CFG_SEQ_Task_LoRaSendOnTxTimerOrButtonEvent), UTIL_SEQ_RFU, SendTxData);
```

Sequencer process must be called within main loop

main.c

```
while (1)
{
  MX_LoRaWAN_Process();
}
```

app_lorawan.c

```
void MX_LoRaWAN_Process(void)
{
  UTIL_SEQ_Run(UTIL_SEQ_DEFAULT);
}
```

Hands-on: Sequencer

Sequencer IDLE task

When nothing to do, sequencer executes IDLE task and enters low-power mode defined by Low Power Manager (STOP2)

sys_app.c

```
void UTIL_SEQ_Idle(void)
{
  UTIL_LPM_EnterLowPower();
}
```

life.augmented

Hands-on: RF duty cycle limitation

According to ETSI EN300220-1 there is a duty cycle limitation in ISM band: “*In a period of 1 hour the duty cycle shall not exceed the spectrum access and mitigation requirement values…*”. For 868MHz sub-band used by Lora it is **1%**.

Lorawan protocol stack automatically handles the RF duty cycle feature for EU868 region. See file RegionEU868.h for relevant definition

```
#define EU868_DUTY_CYCLE_ENABLED  1
```

It is practical approach to disable RF duty cycle feature during development

```
LmHandlerSetDutyCycleEnable(false);
```

life.augmented

Let’s code...

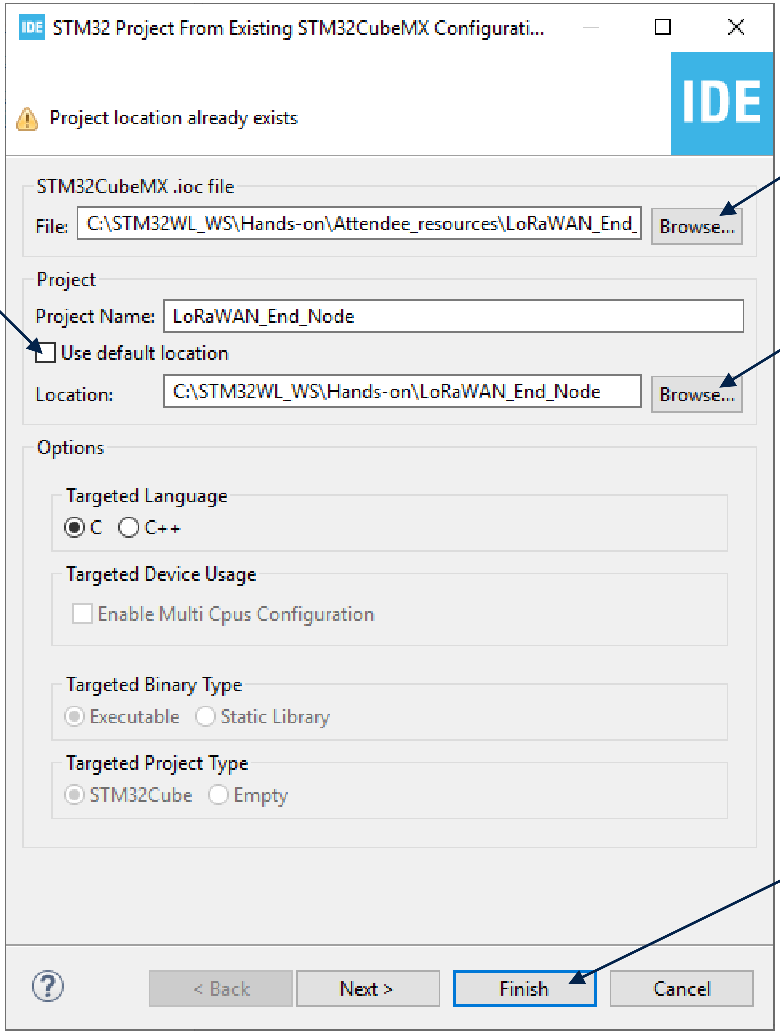
As we are limited in time and we will use pre-configured STM32CubeMX project …\STM32WL_WS\Hands-on\Attendee_resources**LoRaWAN_End_Node.ioc**

Project configuration file .ioc can be also found within STM32CubeWL repository …\STM32Cube\Repository\STM32Cube_FW_WL_V1.0.0\Projects\NUCLEO-WL55JC\Applications\LoRaWAN\LoRaWAN_End_Node\LoRaWAN_End_Node.ioc

Hands-on

1. Open STM32CubeIDE v1.5.0

2. File → New → STM32 Project from an Existing STM32CubeMX Configuration File (.ioc)



...\STM32WL_WS\Hands-on\ Attendee_resources**LoRaWAN_End_Node.ioc**

Uncheck

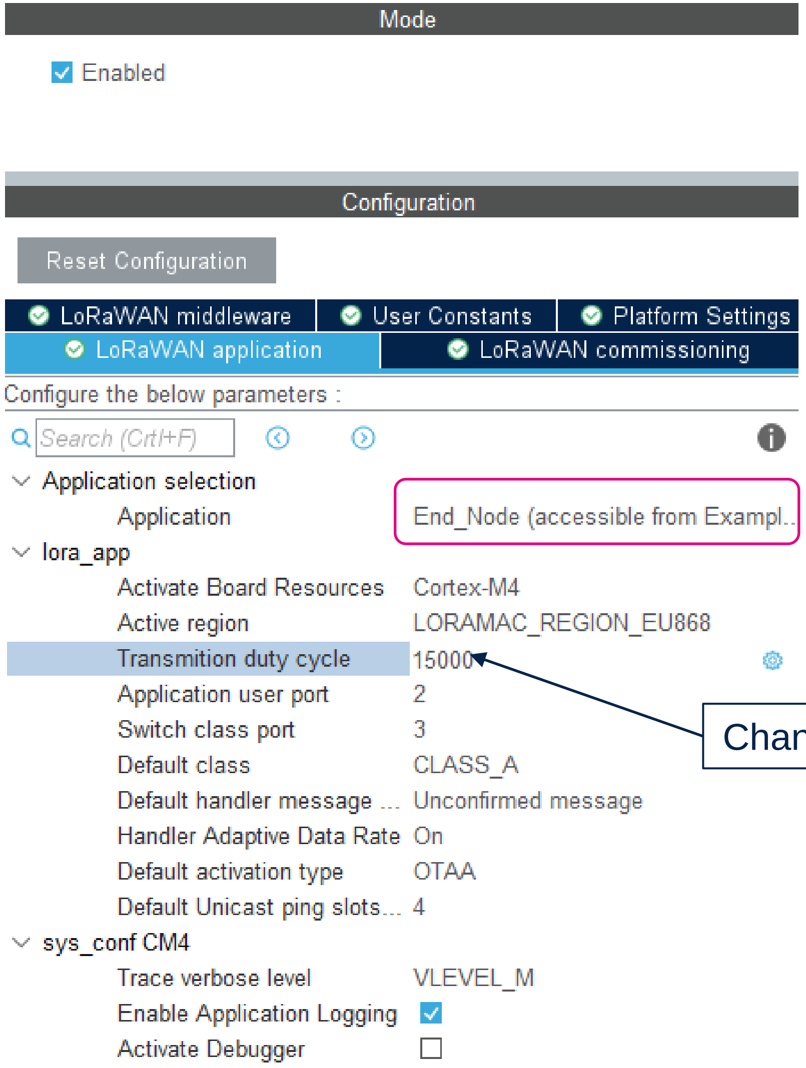
...\STM32WL_WS\Hands-on\ **LoRaWAN_End_Node**

Press when finished, wait until project creation

life.augmented

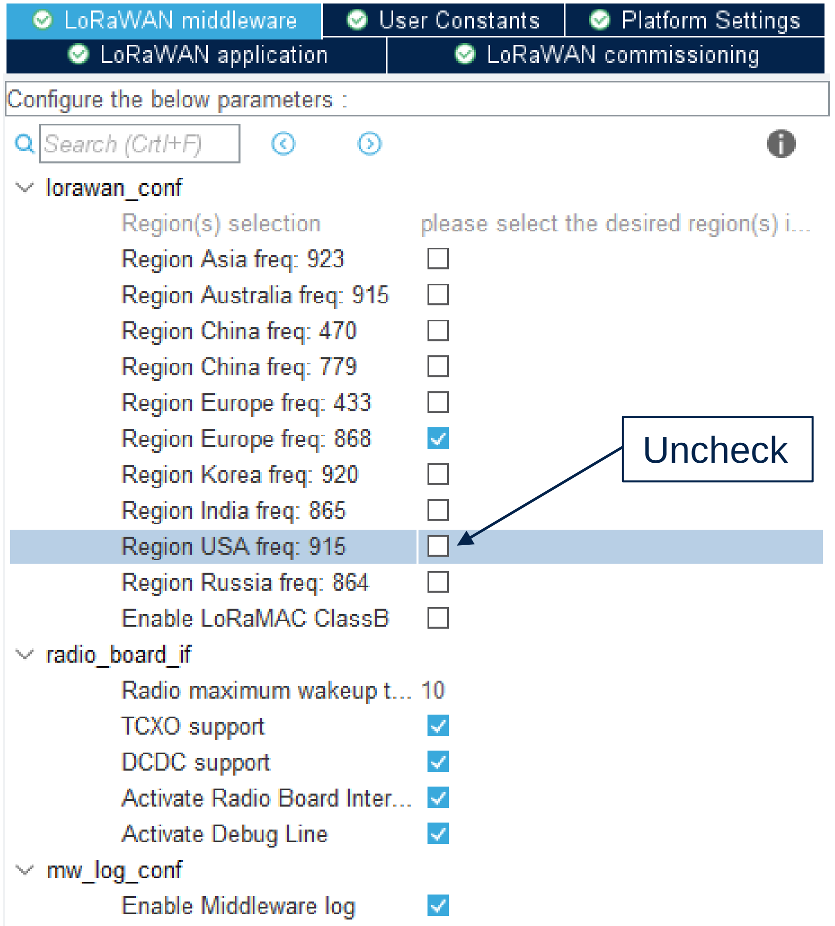
Pinout & Configuration → Middleware → LORAWAN → LoRaWAN application

Hands-on



Change to 15000 (15s)

... → LoRaWAN middleware



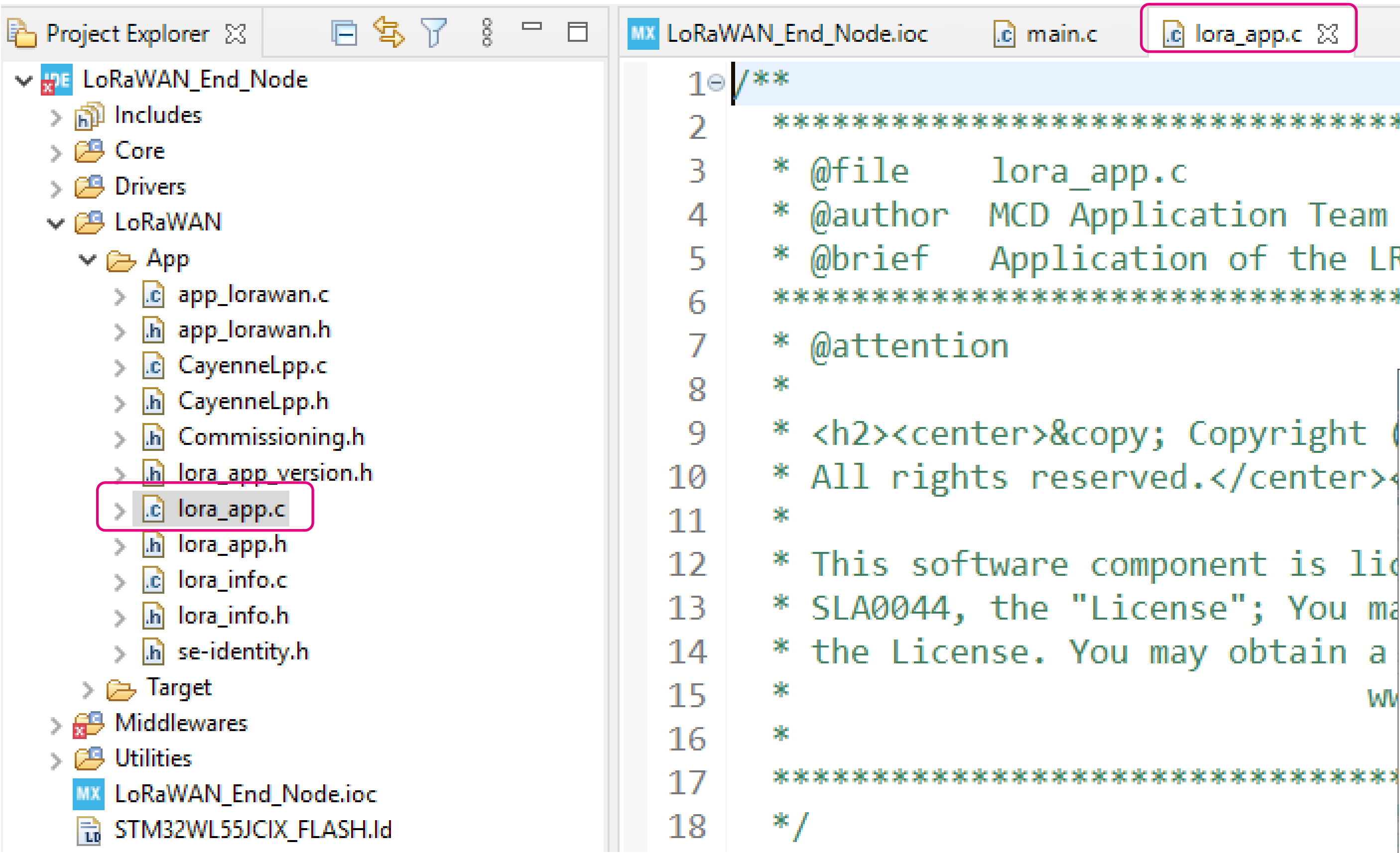
Uncheck

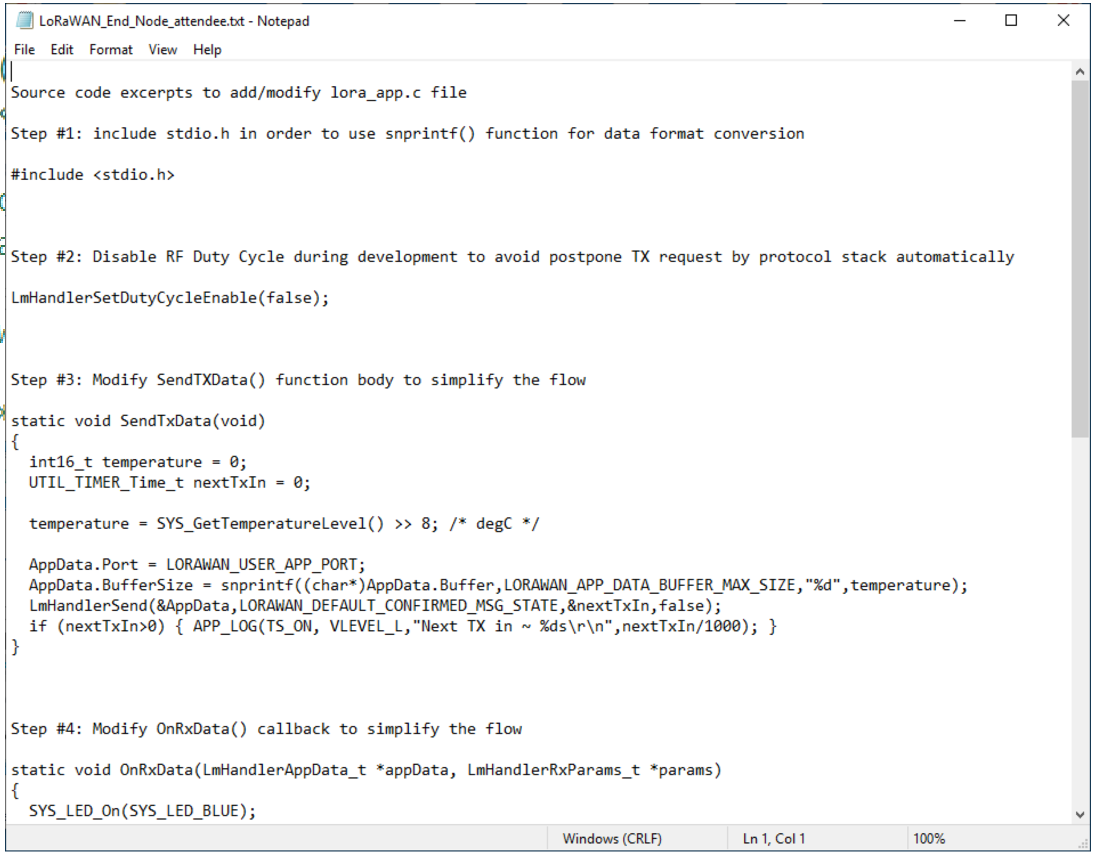
Generate source code

life.augmented

Hands-on

1. Open in STM32CubeIDE editor file **lora_app.c**





2. Open in text editor file ...\STM32WL_WS\Hands-on \Attendee_resources\ **LoRaWAN_End_Node_attendee.txt**

life.augmented

Modify lora_app.c

Step #1: include stdio.h as snprintf() is needed to convert data format and copy to the RAM buffer

```c
/* USER CODE BEGIN Includes */
#include <stdio.h>
/* USER CODE END Includes */
```

Step #2: Disable RF Duty Cycle during development to avoid postpone TX request by protocol stack automatically

```c
/* USER CODE BEGIN LoRaWAN_Init_Last */
  LmHandlerSetDutyCycleEnable(false);
/* USER CODE END LoRaWAN_Init_Last */
```

life.augmented

Modify lora_app.c

Step #3: modify SendTXData() function body to simplify the flow, send temperature sensor value to the network only, overwrite the previous one with new function body from file LoRaWAN_End_Node_attendee.txt (copy/paste)

```
static void SendTxData(void)
{
  int16_t temperature = 0;
  UTIL_TIMER_Time_t nextTxIn = 0;

  temperature = SYS_GetTemperatureLevel() >> 8; /* degC */

  AppData.Port = LORAWAN_USER_APP_PORT;
  AppData.BufferSize = snprintf((char*)AppData.Buffer,LORAWAN_APP_DATA_BUFFER_MAX_SIZE,"%d",temperature);
  LmHandlerSend(&AppData,LORAWAN_DEFAULT_CONFIRMED_MSG_STATE,&nextTxIn,false);
  if (nextTxIn>0) { APP_LOG(TS_ON, VLEVEL_L,"Next TX in ~ %ds\r\n",nextTxIn/1000); }
}
```

Explanation of **SendTxData()**

1. MCU Temperature Sensor variable

```
int16_t temperature = 0;
```

2. RF duty cycle variable to return the app layer next TX in

```
UTIL_TIMER_Time_t nextTxIn = 0;
```

3. Get MCU junction Temperature Sensor, convert fixed point q8.7 to integer

```
temperature = SYS_GetTemperatureLevel() >> 8; /* degC */
```

4. Set LoRaWAN application port number

```
AppData.Port = LORAWAN_USER_APP_PORT;
```

5. Update TX buffer, convert integer format to ASCII

```
AppData.BufferSize = snprintf((char*)AppData.Buffer,LORAWAN_APP_DATA_BUFFER_MAX_SIZE,"%d",temperature);
```

6. TX data to the network

```
LmHandlerSend(&AppData,LORAWAN_DEFAULT_CONFIRMED_MSG_STATE,&nextTxIn,false);
```

7. Option: log RF duty cycle exceeded event

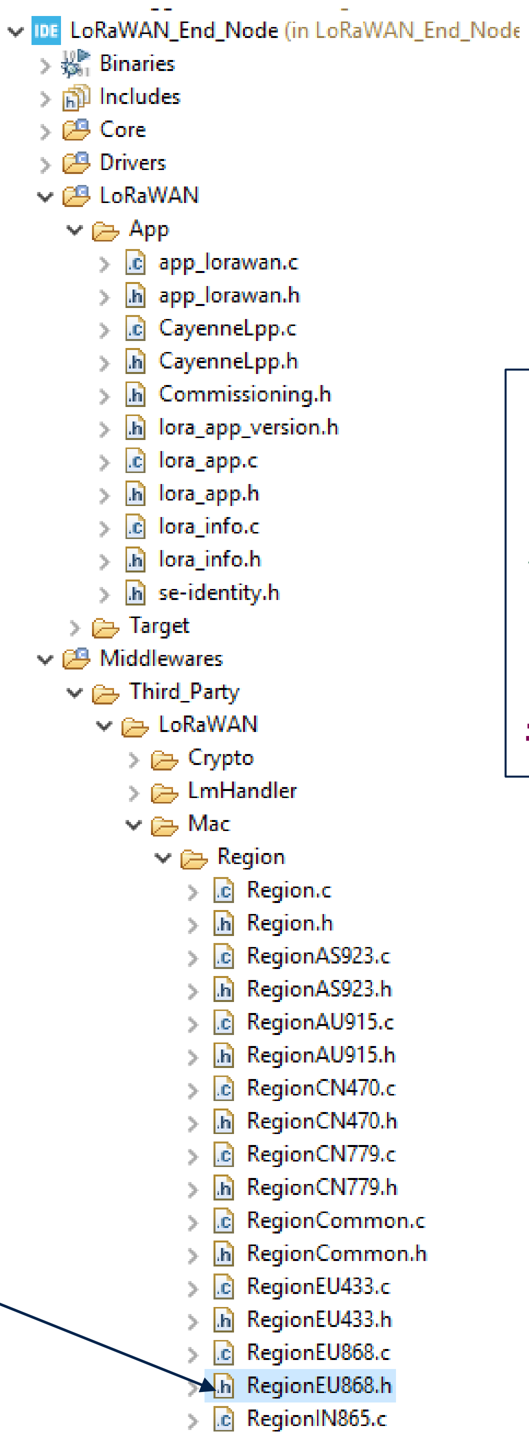
```
if (nextTxIn>0) { APP_LOG(TS_ON, VLEVEL_L,"Next TX in ~ %ds\r\n",nextTxIn/1000); }
```

Modify RegionEU868.h

According to UM2587, P-NUCLEO-LRWAN gateway supports 8 Lora BW=125kHz channels

CH	EU868
0	867.1
1	867.3
2	867.5
3	867.7
4	867.9
5	868.1
6	868.3
7	868.5

Regional settings can be modified accordingly, file: **RegionEU868.h**




```
•
•
/*!
 * LoRaMac maximum number of channels
 */
#define EU868_MAX_NB_CHANNELS  8  //16
```

Our application is ready to send data and we could finish here but let’s implement **RX** process

Modify lora_app.c

Step #4: modify OnRxData() callback to simplify the flow, analyze received 1-byte payload buffer and print out relevant log message

```
static void OnRxData(LmHandlerAppData_t *appData, LmHandlerRxParams_t *params)
{
  SYS_LED_On(SYS_LED_BLUE);
  UTIL_TIMER_Start(&RxLedTimer);
  switch (appData->Port)
  {
    case LORAWAN_USER_APP_PORT:
      switch (appData->Buffer[0])
      {
        case 'I':
          APP_LOG(TS_ON, VLEVEL_L,"AppMsg -> INCREASE temperature.\r\n\r\n");
          break;
        case 'D':
          APP_LOG(TS_ON, VLEVEL_L,"AppMsg -> DECREASE temperature.\r\n\r\n");
          break;
        case 'E':
          APP_LOG(TS_ON, VLEVEL_L,"AppMsg -> EQUALIZED temperature.\r\n\r\n");
          break;
        default:
          APP_LOG(TS_ON, VLEVEL_L,"AppMsg -> ERROR: received data inconsistent.\r\n\r\n");
          break;
      }
      break;
    default:
      break;
  }
}
```

life.augmented

Explanation of OnRxData()

1. Switch on blue LED

```
SYS_LED_On(SYS_LED_BLUE);
```

2. Start RX LED sw timer

```
UTIL_TIMER_Start(&RxLedTimer);
```

3. Validate the received packet port number

```
switch (appData->Port)
```

4. Analyze received 1-byte payload buffer and perform action accordingly

```
switch (appData->Buffer[0])
```

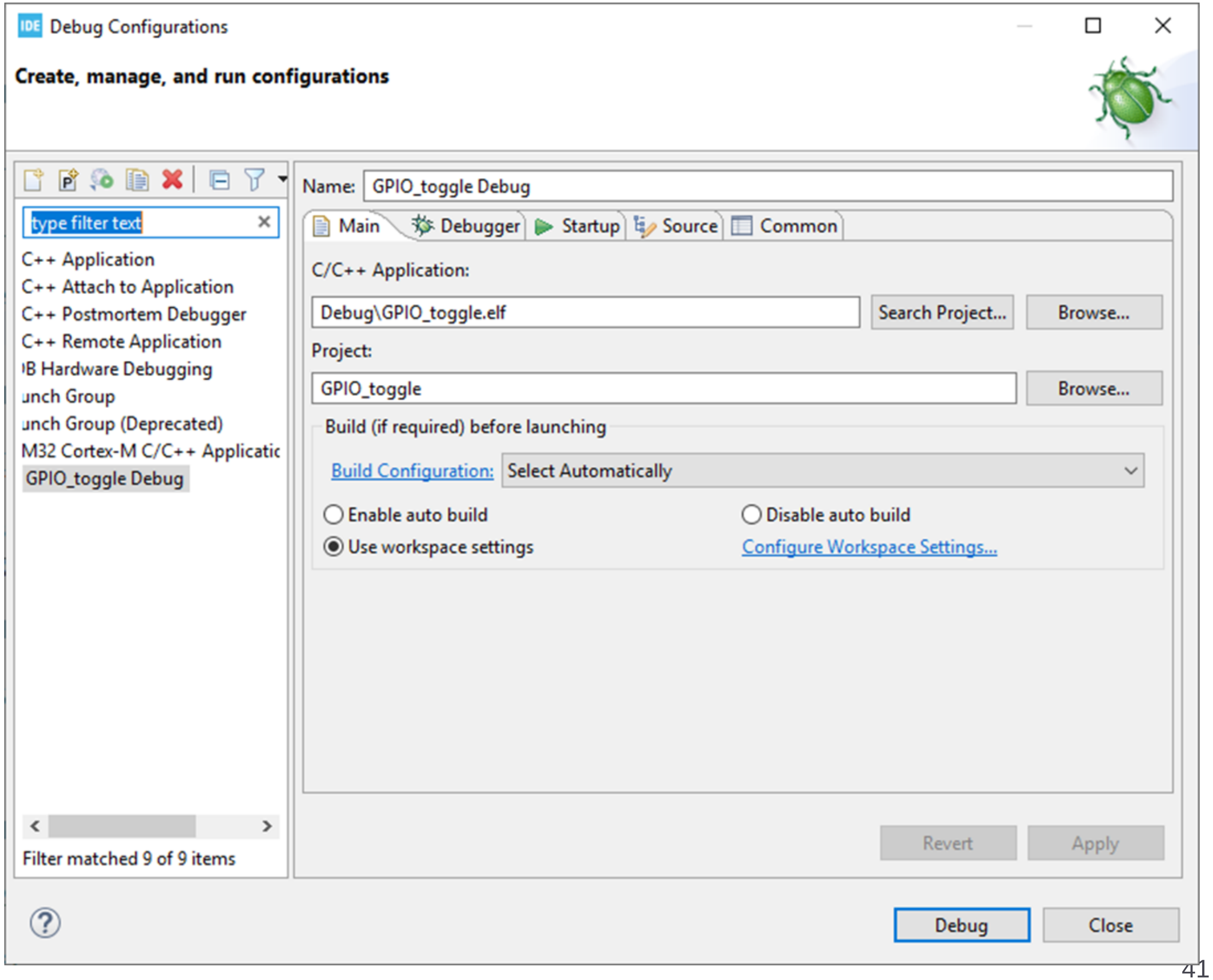
life.augmented

Hands-on: build & flash

Build

Debug

Stop debug when MCU is flashed



Modify lora_app.c

Step #5 (Option): following build console output, comment out not used variable `AppLedStateOn` to avoid compiler warnings, then build project again

Private variables section

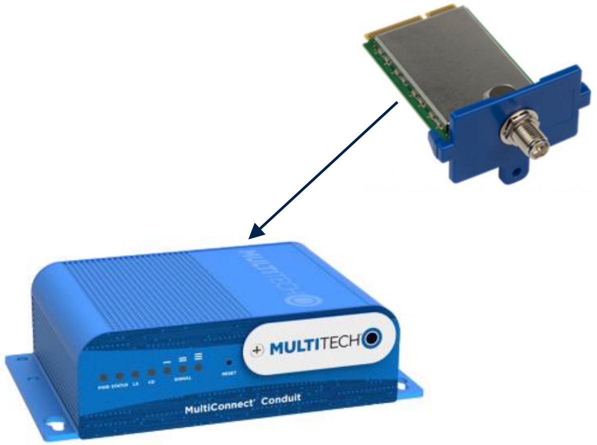
```
.
.
/**
  * @brief Specifies the state of the application LED
  */
//static uint8_t AppLedStateOn = RESET;
.
.
```

life.augmented

Hands-on

All in one and portable LoRaWAN infrastructure to build network: gateway + network server + application server in one box

One of possible solution: Multitech MultiConnect Conduit,
- embeds both LoRa gateway, network server and application server (Node-RED)
- TCP/IP connection via Eth/WiFi or GSM,
- WEB user interface



No need for commissioning each new joining node due to practical reason

The equivalent of above one is setup for P-NUCLEO-LRWAN2 gateway and PC described in workshop prerequisites.

life.augmented

Network server and keys configuration

Channel Plan

Channel Plan	EU868	Duty Cycle Period	60 min
Channel Mask	Edit	Additional Channels 1	869.5 MHz

Network

Network Mode	Public LoRaWAN	Lease Time	00-00-00
Join Delay (sec)	5	Address Range Start	00:00:00:01
Rx1 Delay (sec)	1	Address Range End	FF:FF:FF:FE
NetID	000000	Queue Size	1

Settings

Tx Power	10 dBm	ADR Step	30 cBm
Antenna Gain	0 dBi	Min Datarate	0 - SF12BW125
Rx 1 DR Offset	0	Max Datarate	5 - SF7BW125
Rx 2 Datarate	0 - SF12BW125	ACK Timeout	5000

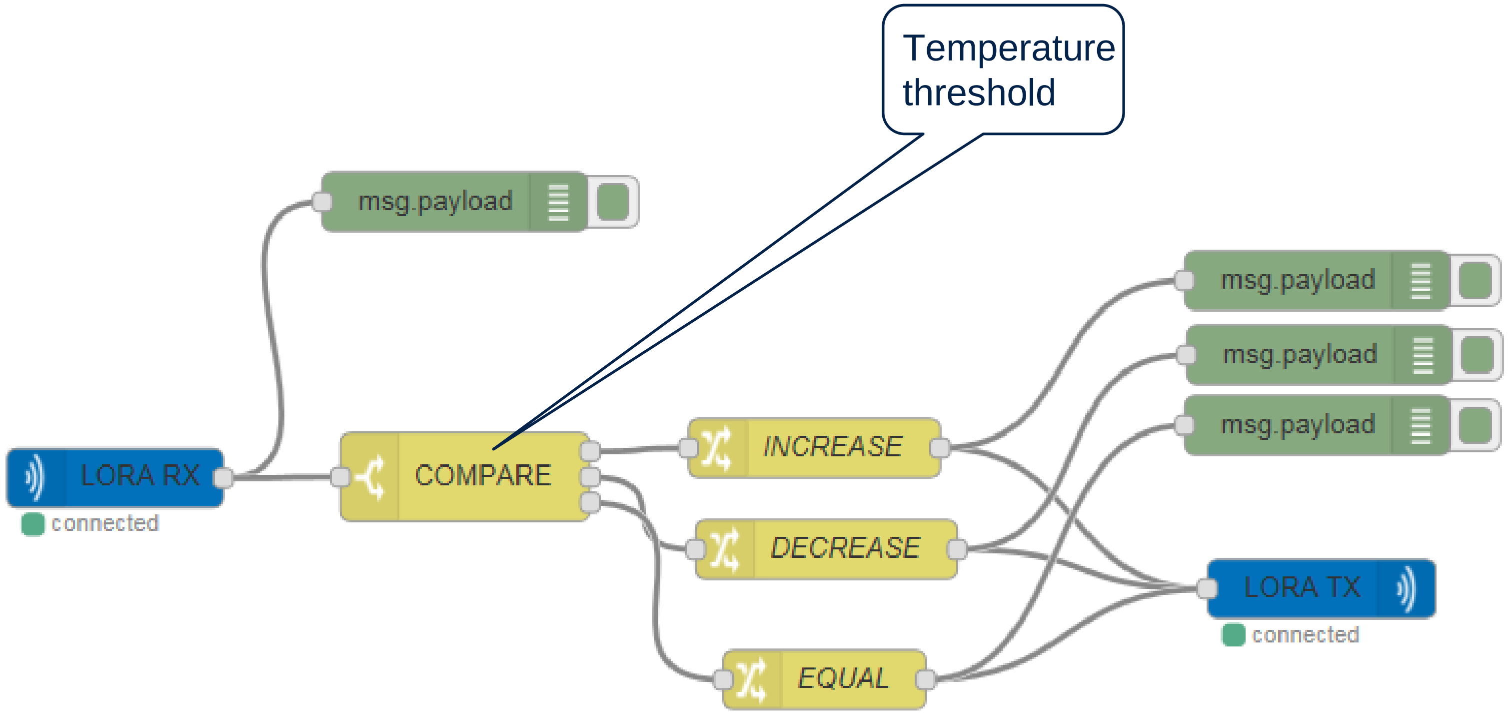
Database

Database Path	/var/config/lora/lora-ne	Reduce Uplink Writes	☐
Backup Interval	3600	Skip Field Check	☐
Trim Interval	600	Trim Rows	100

Local Network Settings

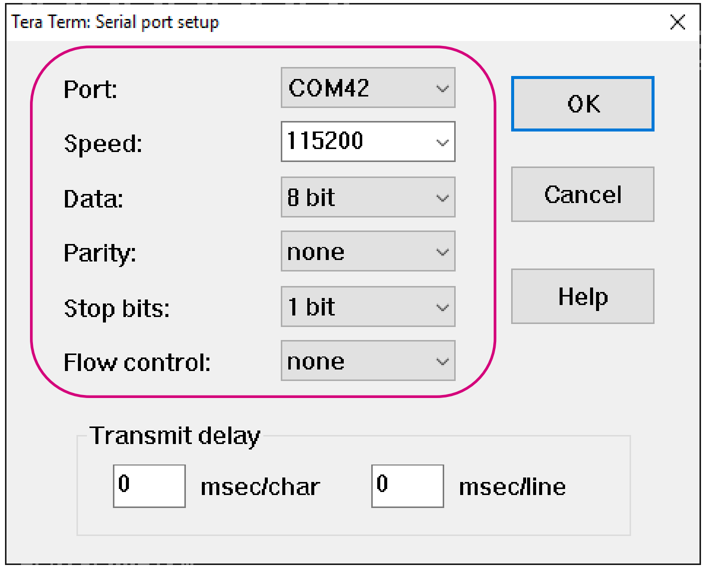
Enabled	☑
Network ID (AppEUI)	EUI
EUI	01:01:01:01:01:01:01:01
Network Key (AppKey)	Key
Key	2B:7E:15:16:28:AE:D2:A6:AB:F7:15:88:09:CF:4F:3C
Default Profile	DEFAULT-CLASS-A

Application server functionality (Node-RED)

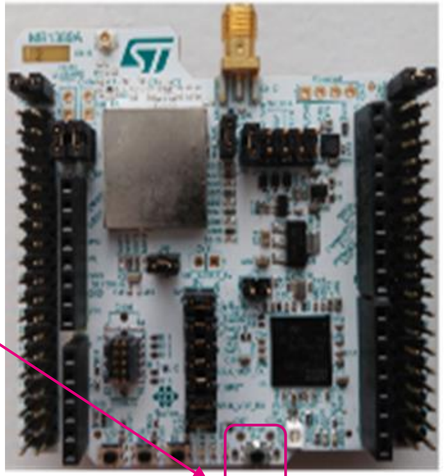


Hands-on: terminal log

1. Start Nucleo VCP terminal session: 115200,8,N,1



2. Reset MCU



3. Wait for “JOINED” log message

```
0s069:temp= 23
30s072:TX on freq 869000000 Hz at DR 0
31s237:MAC txDone
32s259:RX_1 on freq 869000000 Hz at DR 0
32s396:PRE OK
32s931:HDR OK
33s422:MAC rxDone

###### ========== MCPS-Confirm =============
33s425:AppMsg -> INCREASE temperature.

0s000:MAC_VERSION= V4.4.3_rc0
###### DevEui:    01-50-36-32-77-33-4A-20
###### AppEui:    01-01-01-01-01-01-01-01
###### AppKey:    2B 7E 15 16 28 AE D2 A6 AB F7 15 88 09 CF 4F 3C
###### GenAppKey: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0s057:TX on freq 868100000 Hz at DR 0
1s550:MAC txDone
6s573:RX_1 on freq 868100000 Hz at DR 0
6s709:PRE OK
7s244:HDR OK
8s391:MAC rxDone

###### = JOINED = OTAA =====================
```

life.augmented

Hands-on: terminal log

4. When node is joined, wait `APP_TX_DUTYCYCLE` [ms] for TX

```
###### = JOINED = OTAA =====================
30s067:temp= 24
30s070:TX on freq 869000000 Hz at DR 0
31s235:MAC txDone
32s257:RX_1 on freq 869000000 Hz at DR 0
32s394:PRE OK
32s929:HDR OK
33s420:MAC rxDone

###### ========== MCPS-Confirm =============
33s423:AppMsg -> INCREASE temperature.
```

5. When data has been sent, wait for AS response

life.augmented

Hands-on: terminal log

App behavior when RF duty cycle is enabled, see #7 (option) of function **`SendTxData()`**

RF duty cycle is exceeded

Next TX in

```
960s072:temp= 25
960s072:Next TX in ~ 84s
970s072:temp= 25
970s072:Next TX in ~ 74s
980s072:temp= 25
980s072:Next TX in ~ 64s
990s072:temp= 23
990s072:Next TX in ~ 54s
1000s072:temp= 25
1000s072:Next TX in ~ 44s
1010s072:temp= 25
1010s072:Next TX in ~ 34s
1020s072:temp= 25
1020s072:Next TX in ~ 24s
1030s072:temp= 25
1030s072:Next TX in ~ 14s
1040s072:temp= 25
1040s072:Next TX in ~ 4s
1050s072:temp= 23
1050s073:TX on freq 867900000 Hz at DR 0
1051s240:MAC txDone
1052s261:RX_1 on freq 867900000 Hz at DR 0
1052s399:PRE OK
1052s934:HDR OK
1053s425:MAC rxDone
1053s428:AppMsg -> INCREASE temperature.
```

What about current consumption ?

Hands-on: lab way of consumption measurement

Rx/Tx Rx/Tx Rx/Tx

I_{DD} = few hundreds of nA up to few tens of mA

changing rapidly

life.augmented

Hands-on: low cost way **X-NUCLEO-LPM01A**

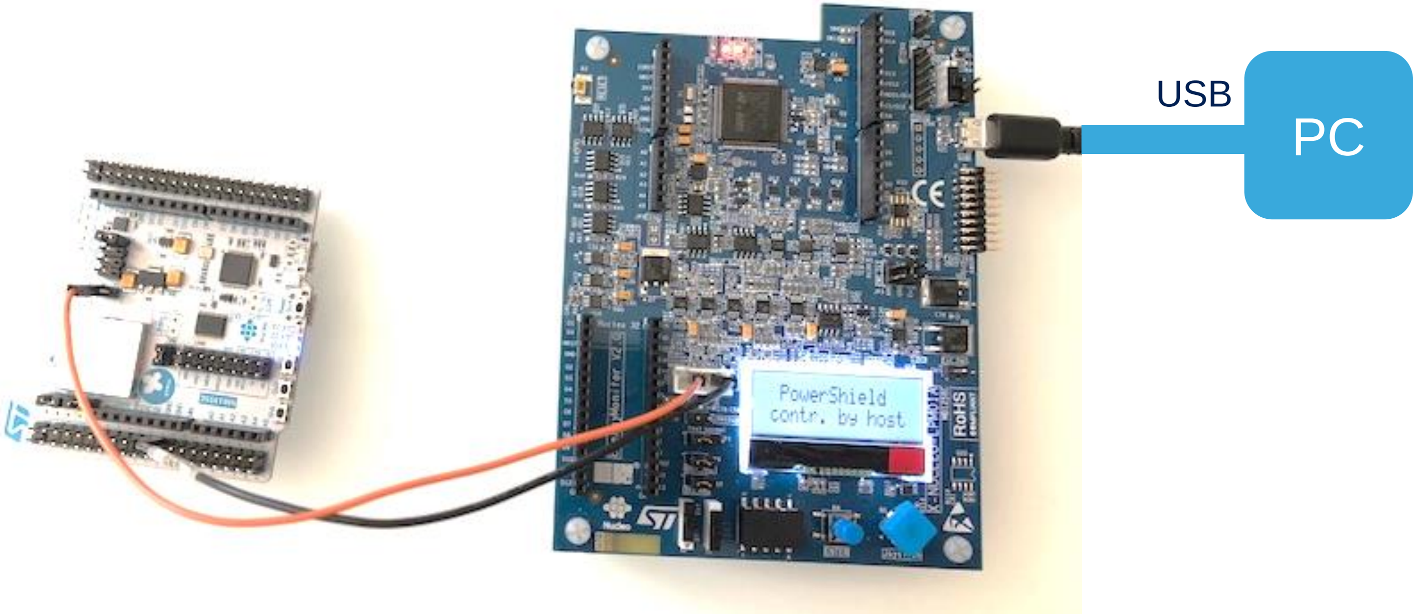
- **100 kHz bandwidth**, 3.2 MS/s sampling rate
- Current from **100 nA to 50 mA**
- Power measurement from 180 nW to 165 mW
- Energy measurement computation by power measurement time integration
- Execution of EEMBC ULPMark™ tests

STM32Cube Monitor-Power

EMBC® EMBEDDED MICROPROCESSOR BENCHMARK CONSORTIUM

ULPMark™ An EEMBC Benchmark

IoTMark™-BLE An EEMBC Benchmark

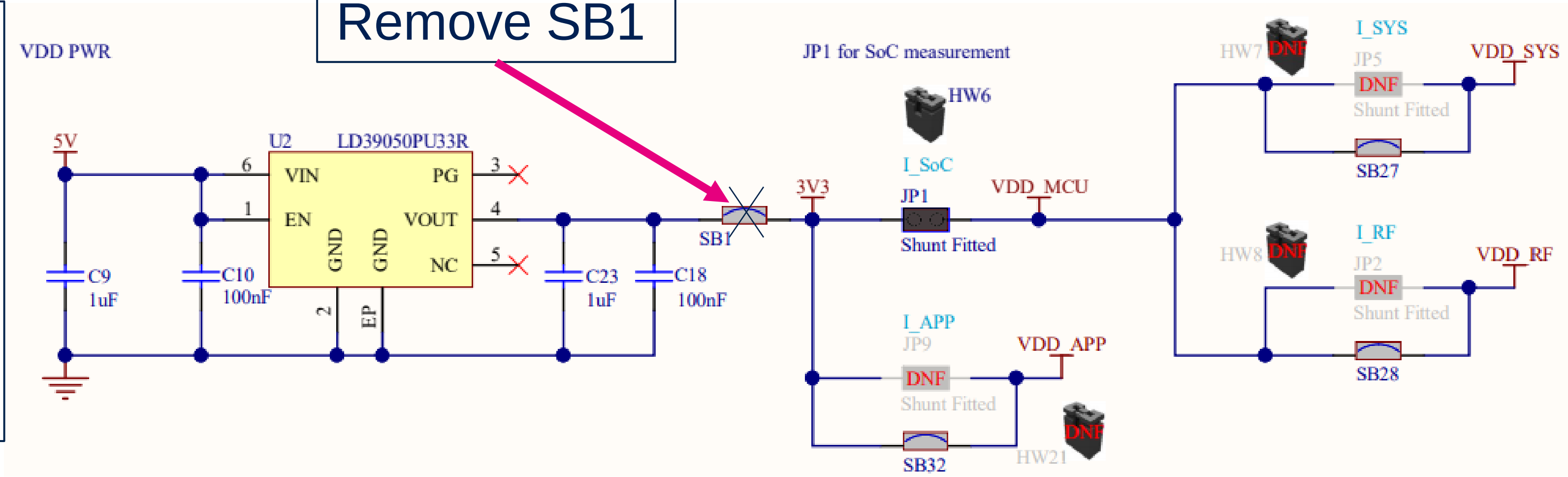


https://www.st.com/en/development-tools/stm32cubemonpwr.html

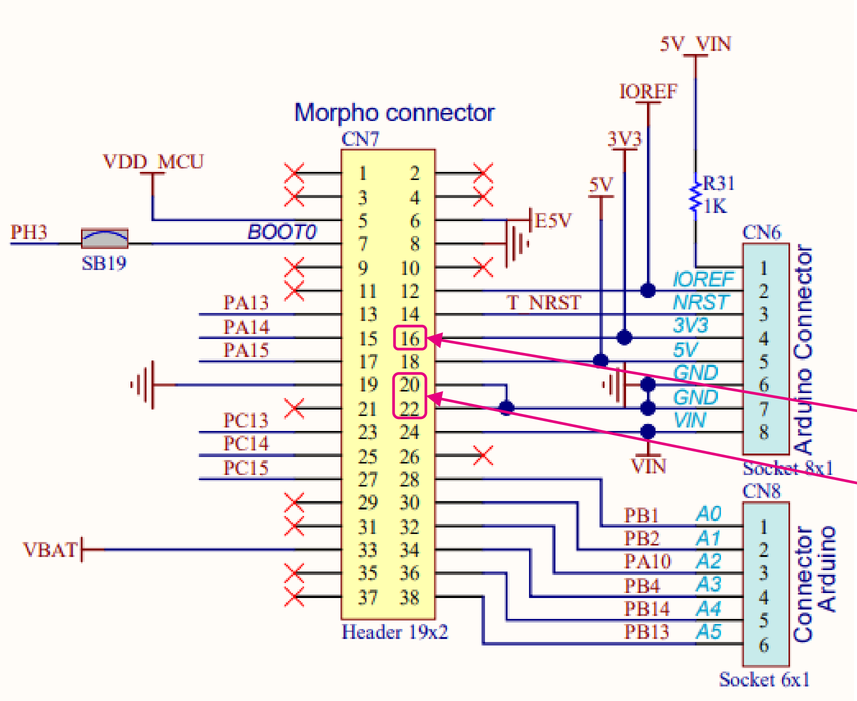
Hands-on: Nucleo MB1389C & X-NUCLEO-LPM01A

HW modification of NUCLEO MB1389C to avoid parasitic current flow and measure current consumption including both SoC and external circuits (antenna switch etc.) covering real application.

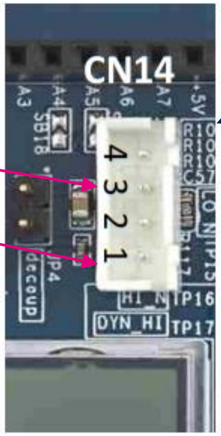
Remove SB1



Connection of X-NUCLEO-LPM01A to supply NUCLEO MB1389C



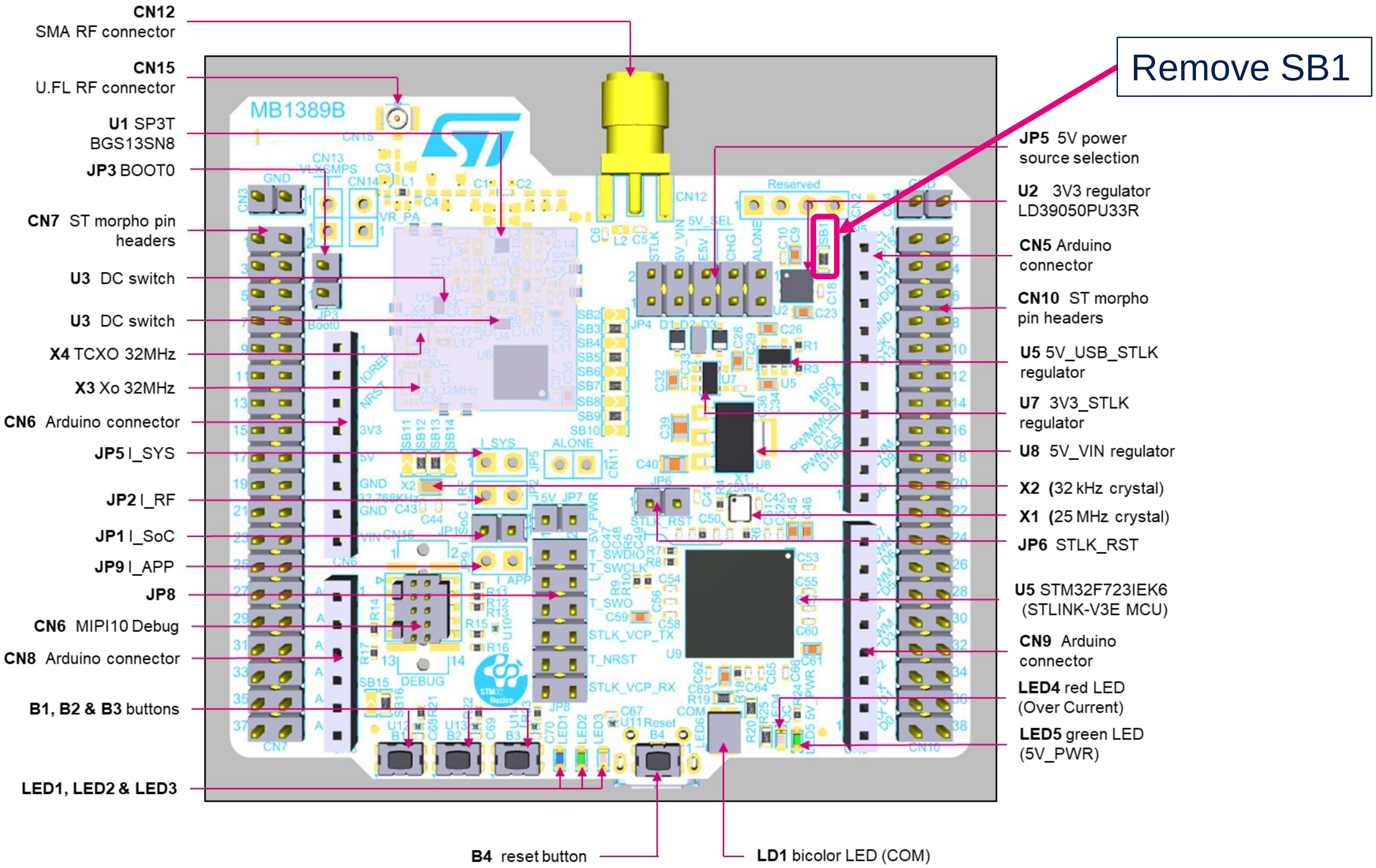
CN14 pin	Signal	Usage
Pin 1	**GND (-)**	Ground of the target
Pin 2	VDD	Alternate power supply source (not measured)
Pin 3	**VOUT** (+)	Positive connection of the target, current is measured
Pin 4	VOUT_MONITORING	Mirror of VOUT. Allows VOUT monitoring without impacting current/power measurements



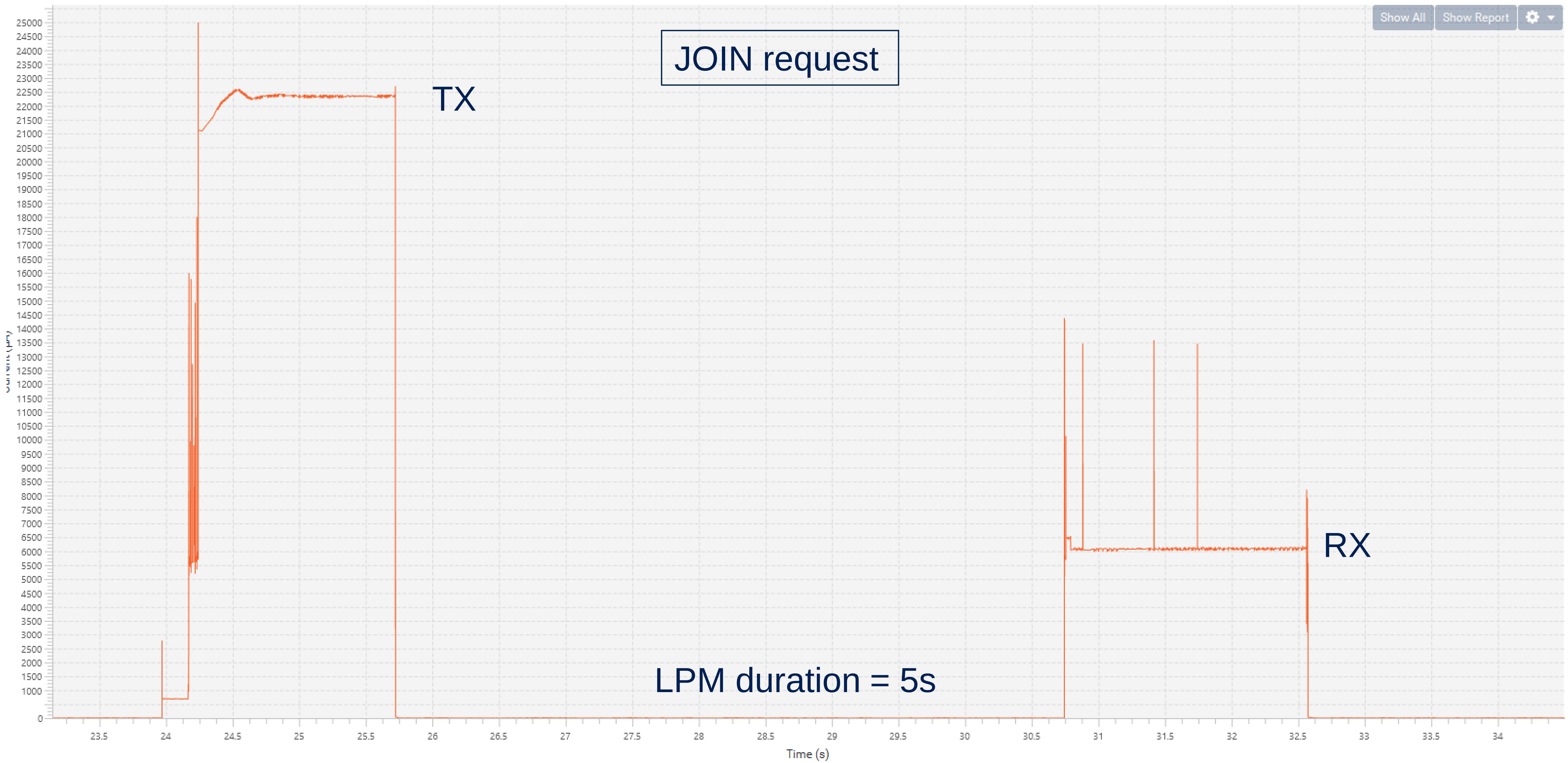
X-NUCLEO-LPM01A Connector CN14

life.augmented

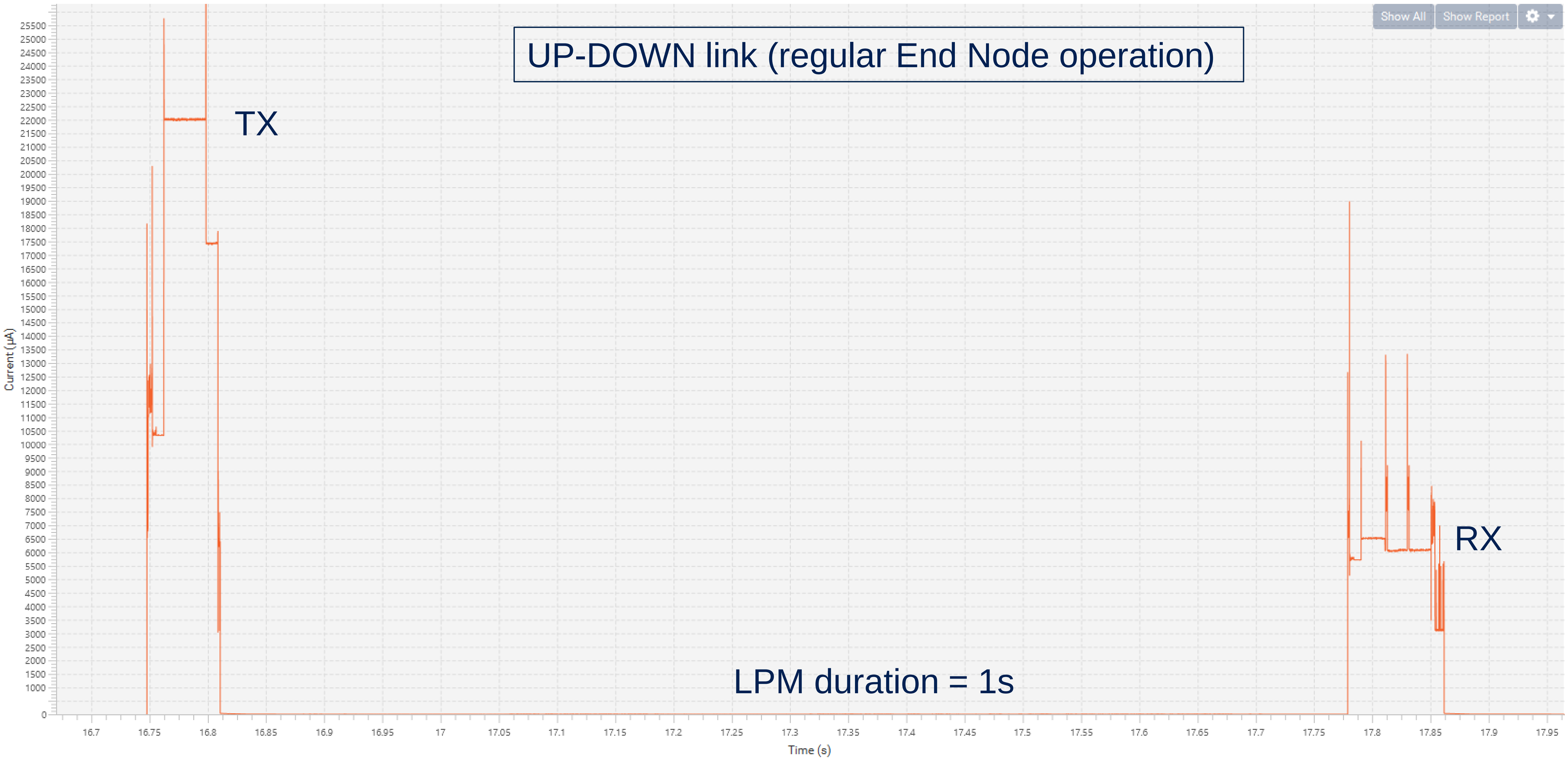
Hands-on: Nucleo MB1389C & X-NUCLEO-LPM01A



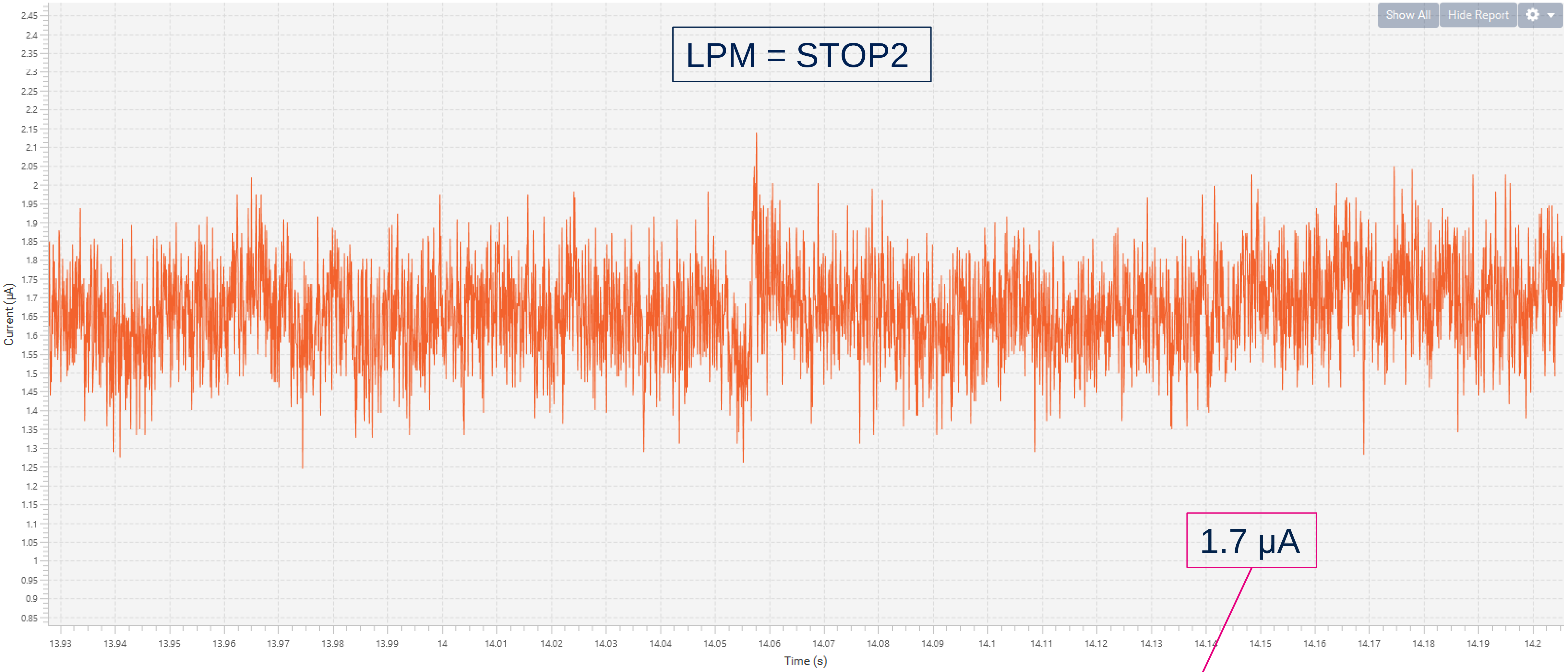
Hands-on: JOIN profile



Hands-on: UP-DOWN link profile



Hands-on: LPM profile



FULL							SELECTED TIME FRAME			0.278 s	
	Min:	Max:	Average:		Energy:		Min:	Max:	Average:		Energy:	
Current:	0.000	24566.650	82.104	µA	4767.968	µJ	1.244	2.138	1.664	µA	1.526	µJ

App configuration & debug

The configuration of app features is possible in file **sys_conf.h**

Adding a Nucleo sensor shield (see comments in file sys_sensor.c, additional effort needed)

```
#define SENSOR_ENABLED  0
```

Trace (log) verbosity level

```
#define VERBOSE_LEVEL     VLEVEL_M
```

Log enable: UART in DMA mode

```
#define APP_LOG_ENABLED   1
```

Debugger mode: enables debugger and 4 debug pins (higher power consumption), see also sys_debug.c/h, 2 debug pins used in radio.c/radio_conf.h to indicate RX/TX mode

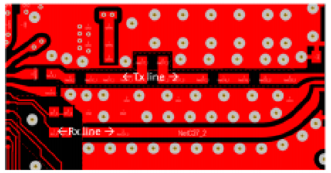
```
#define DEBUGGER_ON       0
```

1. Disable Low Power mode of the app

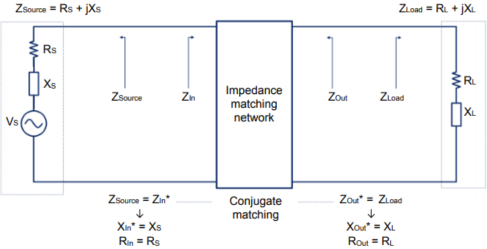
```
#define LOW_POWER_DISABLE 0
```

STM32WL hw design support

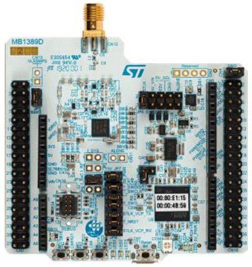
AN5407 “ Optimized RF board layout for STM32WL Series”



AN5457 “RF matching network design guide for STM32WL Series”



STM32WL Reference design: QFN-48, BGA-73, low & high power RF output covered



STM32WL Nucleo-64 board resources

Certification

- LoRaWAN certification ensures interoperability and compliance on any LoRaWAN® network (functional test if node protocol stack and app are compliant with LoRaWAN specification),
- Optional extended RF tests: Total Radiated Power, Total Isotropic Sensitivity to fulfill network service providers minimum RF requirements
- LoRaWAN Certified logo and LoRa Alliance product listing website after successful certification,
- Device manufacturer must be a member of LoRa Alliance to go through certification process and use LoRa Certified logo,
- Currently the certification program is for Class A devices in: EU863-870MHz, US902-928MHz, AS923MHz, KR920-923,IN865-867MHz
- Regulatory testing (CE/FCC) can take place before, after or at the same time with LoRaWAN certification testing,
- Process details (including FAQs): **https://lora-alliance.org/lorawan-certification**

life.augmented

Thank you

life.augmented