LoRaWAN stack

LoRaWAN stack

Version 1.0

Hello, and welcome to this presentation of the LoRa stack.

LoRaWAN

- The LoRaWAN stack used for STM32WL development is LoRaWAN L2 V1.0.3 compatible
- Based on LoRaMac-node from Semtech on github, a pre-release including the L2 1.0.4 implementation has been posted v4.5.0_rc1 on 24th Nov 2020, on *https://github.com/Lora-net/LoRaMac-node/releases*
- Supported features
  - Class A (Unicast)
  - Class C (Unicast and Multicast)
  - Class B (Unicast and Multicast)
- The LoRaWAN L2 V1.0.4 specification is available since October 2020, on *https://lora-alliance.org/resource-hub*
- LoRaWAN L2 V1.0.3 versus V1.0.4
  - V1.0.4 will be a final V1.0.x LoRaWAN specification
  - LoRa Alliance Technical Committee decides to backport some CRs from V1.1 to V1.0.4
  - To have a significant quality rework on V1.0.4
  - In functional point of view there is no difference between these two versions

life.augmented

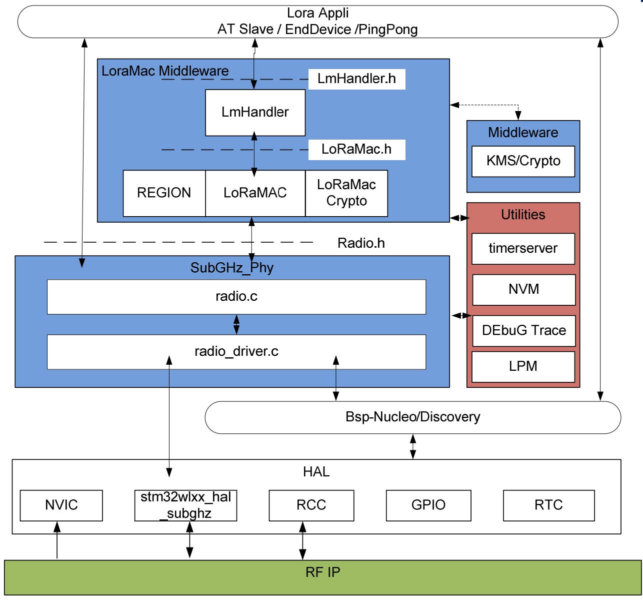
The LoRaWAN embedded in STM32WL is LoRaWAN L2 V1.0.3 compatible.
It is based on the Semtech Github LoRaMac-node: https://github.com/Lora-net/LoRaMac-node/
The supported class are: Class A for Unicast, Class B synchronous transmission (thanks to beacon) for Unicast or Multicast, and Class C for continuous transmission for Unicast or Multicast.
On LoRaMac-node Github, the LoraWAN L2 V1.0.4 specification is planned to be available by end of 2020: a pre-release has been posted v4.5.0_rc1 on 24th Nov 2020
The difference between LoRaWAN L2 V1.0.3 and V1.0.4 specifications is that:

- V1.0.4 is planned to be the last v1.0.x specification
- some CRs from v1.1 have been backported to V1.0.4

to have a significant quality rework
- there’s no difference in terms of functionality between the 2 versions

Overview

- STM32WL FW contains:
  - End-Node application
  - AT-Slave application
  - Ping-Pong application
- Middleware
  - LoRaWAN contains the Mac layer
  - SubGHz_Phy contains the Phy Layer
    - Note: i-cube-lrwan will be updated accordingly
  - when LORAWAN_KMS switch is ON in Dual Core
    - In LORAWAN_KMS in CM0PLUS\LoRaWAN\Target\lorawan_conf.h
  - Generic Utilities which are common with STM32WB
- BSP
  - Drives the RF switch, board configuration



STM32WL Firmware contains three different applications : AT_Slave, End_node, PingPong

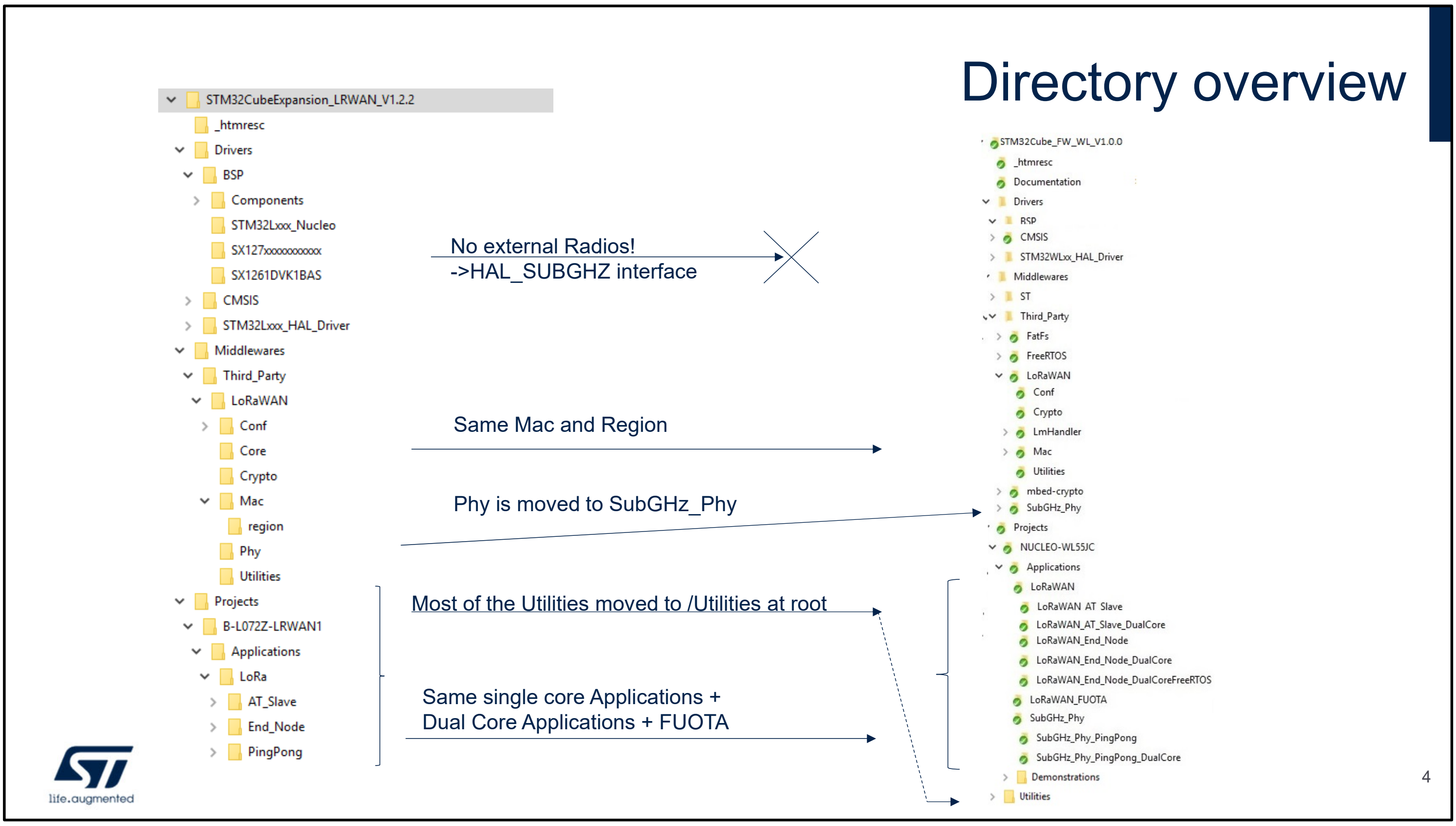
The picture shows the Firmware architecture, the position and interaction of several software layers.

The Middlewares running in the application are:

- LoRaWAN which contains the Medium Access Command Layer
- SubGHz_Phy which contains the SubGHz Physical Layer
- Key Management Storage in DualCore, when LORAWAN_KMS switch is ON in lorawan_conf.h file

The Board Support Package is designed to drive the RF switch, TCXO, DCDC configurations.
One should note that due to STM32CubeMX limitation,

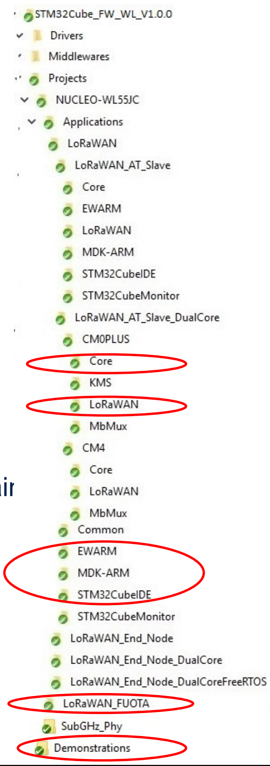
the firmware does not use BSP files but radio_board_if.c/.h for radio related items, and board_resources.c/.h for LED and push buttons



This slide shows the folder structure of the STM32WL FW package on the right in comparison with the Cube Expansion one on the left.
One can notice that in the STM32WL package, BSP isn't composed of external radios, as radio is part of chip, and so it's accessed through HAL_SUBGHZ interface.
LoRAWAN MAC, regions are unchanged but Physical layer is now moved to SubGHz_Phy.
In the STM32WL package, the same Applications as in CubeExpansion are available but also several other applications : the Dual Core version, EndNode with FreeRTOS and also FUOTA applications.

Applications

- In each LoRaWAN application :
  - Any MCU specific files are located in Core
    - peripherals generated by CubeMx: adc/rtc/dma/usart/
    - all peripheral interfaces
  - In LoRaWAN directory are located
    - Applications files in “App”
    - Middleware and board configurations in “Target”
  - 3 toolchains and compilers provided for each application:
    - IAR Embedded Workbench for ARM (EWARM) toolchain
    - RealView Microcontroller Development Kit (MDK-ARM) toolchain
    - STM32CubeIDE
- Package features also
  - FUOTA Application on LoRaWAN
  - Demonstrations: LocalNetwork



life.augmented

5

The Firmware package is composed of several LoRaWAN applications: AT_Slave, End_Node.
Inside each LoRaWAN application, there are several parts:

- MCU specific files containing all peripherals generated by CubeMx and their interfaces, which are in Core directory.
- Application specific files and Middleware and board configurations, which are in LoRaWAN/App and LoRaWAN/Target directories.
- 3 toolchains: IAR(EWARM), MDK_ARM, STM32CubeIDE

The Firmware package is also composed of:

- FUOTA Application on LoRaWAN
- Demonstrations, such as the LocalNetwork example

LoRaWAN middleware

- Interface is updated from LoRaMac.h to LmHandler.h
- LmHandler.c/h is the wrapper to ease application development common for all application.
  - The LmHandler is based on the one from Semtech but updated to enable AT Slave modem
- Regions:
  - RegionAS923
  - RegionAU915
  - RegionCN470
  - RegionCN779
  - RegionEU868
  - RegionEU433
  - RegionKR920
  - RegionRU864
  - RegionUS915
  - RegionIN865

life.augmented

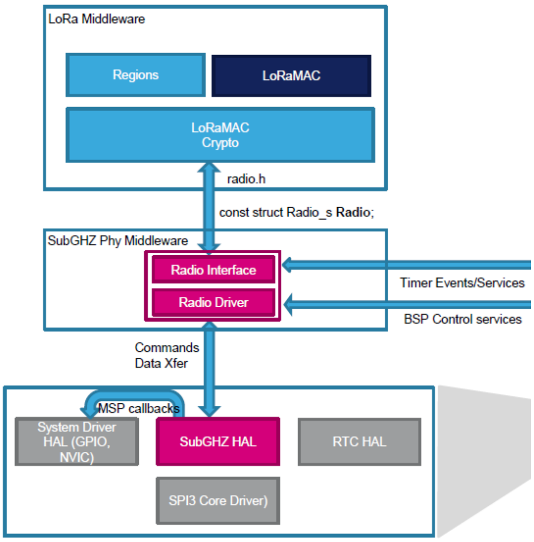
6

The LoRaWAN Middleware interface, which was previously LoRaMac, is updated to LmHandler. LmHandler is the LoRaMac layer interface file implementing a set of APIs to access to the LoRaMAC services. Based initially on Semtech implementation, it has been updated by ST to enable AT Slave modem. Several regions and their corresponding band selection can be selected in the project, you can find the list in this slide.

Note: The current LoRaWAN stack is LoRaWAN Regional Parameters (RP) v1.0.3 compatible

SubGHz_Phy middleware (1/2)

- This diagram shows the interaction model of the LoRa middleware with common SubGHz Phy layer and the low-level drivers



life.augmented

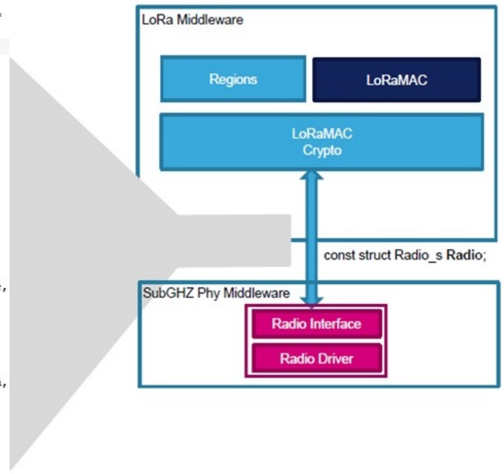
7

The Interaction model displayed here shows how the several software layers communicate together from peripheral HAL and SubGHz HAL to LoRAWAN middleware through SubGHz_Phy Middleware.
The radio interface of SubGHz_Phy Middleware is configured by commands coming from the SubGHz HAL and also by timers A sequencer is triggering radio tasks.

SubGHz_Phy middleware (2/2)

- The LoraWAN Middleware interacts with the Radio RF through the Radio_s structure

```
const struct Radio_s Radio =
{
    RadioInit,
    RadioGetStatus,
    RadioSetModem,
    RadioSetChannel,
    RadioIsChannelFree,
    RadioRandom,
    RadioSetRxConfig,
    RadioSetTxConfig,
    RadioCheckRfFrequency,
    RadioTimeOnAir,
    RadioSend,
    RadioSleep,
    RadioStandby,
    RadioRx,
    RadioStartCad,
    RadioSetTxContinuousWave,
    RadioRssi,
    RadioWrite,
    RadioRead,
    RadioWriteRegisters,
    RadioReadRegisters,
    RadioSetMaxPayloadLength,
    RadioSetPublicNetwork,
    RadioGetWakeupTime,
    RadioIrqProcess,
    RadioRxBoosted,
    RadioSetRxDutyCycle,
    RadioTxPrbs,
    RadioTxCw,
    RadioSetRxGenericConfig,
    RadioSetTxGenericConfig,
};
```



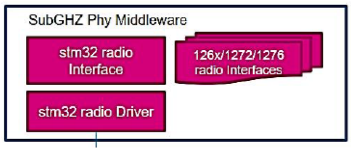
life.augmented

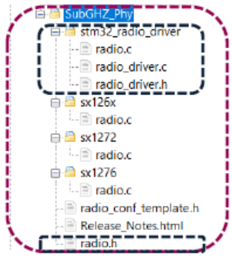
8

Here the focus is on the upper layer part of the interaction model :
The LoraWAN Middleware sends commands to the Radio interface of SubGhz Phy middleware through the Radio_s structure.

Radio interface

- radio.h
  - remains the interface for the loRaMac layer or Application
  - Is the same radio.h as the i-cube-lrwan
- Radio is split into 2 levels
  - Radio high level
    - radio.c (comparable with radio.c driving sx126x)
  - Radio low level functions
    - radio_driver.c is comparable with sx126x.c
    - Interfaces
      - with stm32wlxx_hal_subg.h for SubGHz Ip services
      - With stm32wlxx_nucleo.h for RF BSP services
- Advantage
  - radio.c and radio_driver.c remains comparable with legacy. (Easy to update changes from Semtech)





life.augmented

Let’s detail the SubGHz Phy Middleware.
The interface of radio commands for the LoRaMAC layer or Application is radio.h. This file includes Radio_s structure showed in previous slide and is the same as in i-cube-lrwan.
In STM32WL, Radio is split into 2 different levels:

- radio.c : the radio high level part, comparable with radio.c from Semtech sx126x driver
- radio_driver.c: the radio low level part, comparable with sx126x.c from Semtech. It’s including HAL_SubGHz and BSP nucleo services, as interfaces.

This implementation separation has been done to ease the inclusion of Semtech changes.

Example for transmitting 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 Configuration 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 );
```

life.augmented

10

This is an example of radio interface to setup a transmission (TX):

- First, RadioEvents are initialized with the corresponding callbacks and the radio initialization is called
- Then, the radio configuration is set: In this case, TX in LoRa mode
- The frequency is set
- Finally, the data is sent

Example for receiving 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 a buffer
Radio.Rx( RX_TIMEOUT_VALUE );
```

life.augmented

This is an example of radio interface to setup a reception (RX):
First, RadioEvents are initialized with the corresponding callbacks and the radio initialization is called.
Then, radio configuration is set : In this case, RX in FSK mode
The frequency is set
Finally, the radio reception is enabled : RX1 and/or RX2 windows are going to be opened.

STM32WLxx HAL_SubGHz

- HAL APIs to access the Radio
  - Internal SPI
  - RCC and NVIC are set in HAL_SUBGHZ_MspInit() function
- The SubGHz commands use defined enumeration structure:
  - SUBGHZ_RadioGetCmd_t: Get commands structure
  - SUBGHZ_RadioSetCmd_t: Set commands structure
- API list
  - HAL_StatusTypeDef HAL_SUBGHZ_Init(SUBGHZ_HandleTypeDef *hsubghz);
  - HAL_StatusTypeDef HAL_SUBGHZ_DeInit(SUBGHZ_HandleTypeDef *hsubghz);
  - void HAL_SUBGHZ_ExecSetCmd(SUBGHZ_HandleTypeDef *hsubghz, SUBGHZ_RadioSetCmd_t command, uint8_t *buffer, uint16_t size);
  - void HAL_SUBGHZ_ExecGetCmd(SUBGHZ_HandleTypeDef *hsubghz, SUBGHZ_RadioGetCmd_t command, uint8_t *buffer, uint16_t size);
  - void HAL_SUBGHZ_WriteRegisters(SUBGHZ_HandleTypeDef *hsubghz, uint16_t address, uint8_t *buffer, uint16_t size);
  - void HAL_SUBGHZ_ReadRegisters(SUBGHZ_HandleTypeDef *hsubghz, uint16_t address, uint8_t *buffer, uint16_t size);
  - void HAL_SUBGHZ_WriteBuffer(SUBGHZ_HandleTypeDef *hsubghz, uint8_t offset, uint8_t *buffer, uint16_t size);
  - void HAL_SUBGHZ_ReadBuffer(SUBGHZ_HandleTypeDef *hsubghz, uint8_t offset, uint8_t *buffer, uint16_t size);

life.augmented

HAL SubGHz layer accesses the radio via an internal SPI Link. The NVIC and RCC are set in the MSP Init.
The SubGHz commands use the SUBGHZ_RadioGetCmd_t and SUBGHZ_RadioSetCmd_t structure.
The API list to access the radio registers and set or get commands is displayed here.

STM32WLxx HAL_SubGHz: radioSetCmd

- void HAL_SUBGHZ_ExecSetCmd(SUBGHZ_HandleTypeDef *hsubghz, SUBGHZ_RadioSetCmd_t command, uint8_t *buffer, uint16_t size);
- Example:

Set_Sleep() command

0	1
Opcode	SleepCfg
w	w

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;
```

life.augmented

13

The function of HAL_SubGHz to set a command into the radio is HAL_SUBGHZ_ExecSetCmd.
The first parameter of the function represents the opcode of the command.

- On the right part of the slide, the list of commands opcode is exposed
- On the left part of the slide, an example of HAL_SUBGHZ_ExecSetCmd() call inside SUBGRF_SetSleep() function to Set RF in sleep mode is presented.

STM32WLxx HAL_SubGHz: radioGetCmd

- void HAL_SUBGHZ_ExecGetCmd(SUBGHZ_HandleTypeDef *hsubghz SUBGHZ_RadioGetCmd_t command, uint8_t *buffer, uint16_t size);
- Example:

```
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;
}

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;
}
```

Get_Status() command

0	1
Opcode	Status[7:0]
w	r

byte 0 bits 7:0 **Opcode**: 0xC0

byte 1 bit 7 Reserved, must be kept at reset value.

bits 6:4 **Status Mode[2:0]** sub-GHz radio operating mode

bits 3:1 **Status CmdStatus[2:0]**: Command status

bit 0 Reserved, must be kept at reset value.

```
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;
```

Get_RssiInst() command

0	1	2
Opcode	Status[7:0]	RssiInst[7:0]
w	r	r

byte 0 bits 7:0 **Opcode**: 0x15

byte 1 bits 7:0 **Status[7:0]**: see *Get_Status() command*

byte 2 bits 7:0 **RssiInst[7:0]**: instantaneous RSSI level at the reception time
Signal power = - RssiInst / 2 (in dBm)

life.augmented

The function of HAL_SubGHz to get a value from radio is HAL_SUBGHZ_ExecGetCmd.
The first parameter of the function represents the opcode of the command.

- On the right part of the slide, the list of get commands opcode is exposed
- On the left part of the slide, an example of HAL_SUBGHZ_ExecGetCmd call to get the radio status and the rssi value is presented.

Basic registers

- Registers can be accessed using
  - void HAL_SUBGHZ_WriteRegisters(SUBGHZ_HandleTypeDef *hsubghz,uint16_t address, uint8_t *buffer, uint16_t size);
  - void HAL_SUBGHZ_ReadRegisters(SUBGHZ_HandleTypeDef *hsubghz, uint16_t address, uint8_t *buffer, uint16_t size);
- Radio Interface are taking care of these registers
- Advanced Registers do exist for custom utilization of the RFIP

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
SUBGHZ_GWHITEINIRL (REG_LR_WHITSEEDBASEADDR_LSB)	1	Set Whitening Seed **WHITEINI[7:0]** for GFSK packet
SUBGHZ_GCRCINIR (REG_LR_CRCSEEDBASEADDR)	2	Set Crc Seed for GFSK packet
SUBGHZ_GCRCPOLR (REG_LR_CRCPOLYBASEADDR)	2	Set Crc Polynomial for GFSK packet
SUBGHZ_GSYNCR (REG_LR_SYNCWORDBASEADDRESS)	8	Set the Sync Word for GFSK
SUBGHZ_LSYNCR (REG_LR_SYNCWORD)	2	LoRa synchronization word (public or private)
SUBGHZ_RNGR (RANDOM_NUMBER_GENERATORBASEADDR)	4	Random generator read value
SUBGHZ_RXGAINCR (REG_RX_GAIN)	1	receiver gain control register. Use for normal or boosted gain
SUBGHZ_PAOCPR (REG_OCP)	1	Set maximum current for Over Current Protection.
SUBGHZ_HSEINTRIMR (REG_XTA_TRIM)	1	Xtal oscillator internal Cap trimming value (xtb also exist)
SUBGHZ_HSEOUTTRIMR (REG_XTB_TRIM)	1	Xtal oscillator out Cap trimming value
SUBGHZ_SMPSC0R (REG_DCC_BUCK_CTRL)	1	SMPS clock detection
SUBGHZ_PCR (REG_DCC_BUCK_SEL_OUT_DRIVE)	1	power-supply current limiter

life.augmented

15

The sub-GHz radio peripheral registers can be accessed via HAL sub-GHz functions HAL_SUBGHZ_WriteRegisters () and HAL_SUBGHZ_ReadRegisters().
The list of sub-GHz radio peripheral registers is displayed in this chart.
You can find all the details in STM32WL Reference Manual.

BSP RF services

- Nucleo BSP includes RF APIs:
  - Board configuration
    - BSP_RADIO_Init(void) / BSP_RADIO_DeInit(void)
    - BSP_RADIO_GetWakeUpTime(void)
      - Returns wake up time in ms
    - BSP_RADIO_GetTxConfig(void)
      - Returns board switch configuration: If only Tx Low Power is cabled or Tx High Power, or Both
    - BSP_RADIO_IsTCXO
      - TCXO configuration : may be present or not on the board
    - BSP_RADIO_IsDCDC
      - DCDC configuration: may be present or not on the board
  - RF switch control
    - BSP_RADIO_ConfigRFSwitch(BSP_RADIO_Switch_TypeDef Config)
      - Allows to control the switch Off, in Rx mode, Tx high power, mode Tx Low Power mode

life.augmented

16

The Board Support Package (BSP) of the STM32WL Nucleo board is using the following APIs to set all the RF board specific configurations:

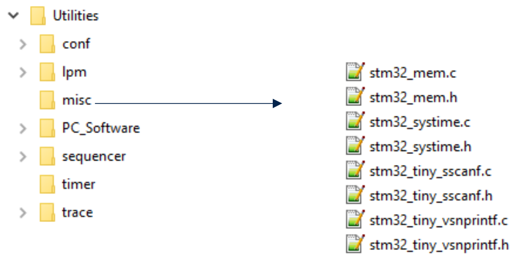
- Wake up time
- Switch configuration
- TCXO
- DCDC
- RF switch mode

These functions are defined in Drivers\BSP\STM32WLxx_Nucleo\stm32wlxx_nucleo_radio.c

Note: *In the current implementation, due to STM32CubeMX limitation, the firmware does not use stm32wlxx_nucleo_radio.c and stm32wlxx_nucleo_radio.c BSP files but radio_board_if.c/.h for radio related items, and* *board_resources.c/.h for LED and push buttons. The choice between the two implementations is done into Core/Inc/platform.h by selecting USE_BSP_DRIVER or MX_BOARD_PSEUDODRIVER*

Embedded utilities

- lpm: low power manager
  - Centralizes low power requirements from module, and go in appropriate low power
    - E.g. when Trace/Dma is printing, MCU shall not go in stop mode2
- sequencer: previously known has scheduler
  - Framework to safely go in low power
  - Records and manages tasks and events with priority handling
- timer: this is the timer list or server
  - Used by middleware and application
- trace: dma trace
  - Uses circular buffer and DMA to print in real time
- misc
  - Systime : set/get the time
  - Tiny_Scanf: low footprint scanf
  - Tiny_printf: low footprint printf



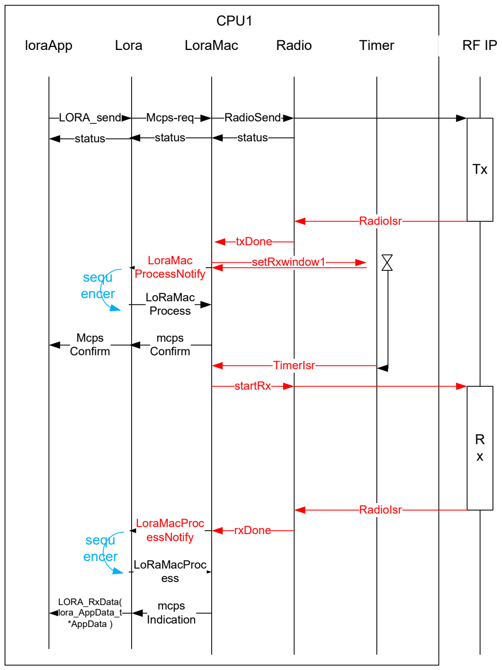
life.augmented

17

The STM32WL FW package embedded utilities are common with other projects.
They are composed of:
Low power manager: which handles low power mode
Sequencer (known previously as scheduler): which schedules all tasks and callbacks according to their appropriate priority level
Timer: which is a common timer list
Trace: which is handling DMA transfer in real time for trace print
Misc: which are miscellaneous modules as systime, tiny scanf and printf.

Dynamic overview

- Class A: Sending frame sequence
- Red arrows are running in interrupt mode
- Sequencer sets a FLAG and starts LoraMacProcess
  - E.g. frame decryption outside IT



life.augmented

18

This message sequence chart shows the messages, flags or callbacks sent between the software layers: Application, MAC, radio and RF IP in order to schedule a Transmission (TX) and open a reception window (RX), in LoRAWAN Class A transmission mode.
Some of the function calls are done in Interrupt mode: they are displayed as red arrows.
The blue arrow represents a call to the sequencer utility, explained in previous slide.