HT MICRON SEMICONDUTORES S.A.
Av. Unisinos, 1550 | 93022-750 | São Leopoldo | RS | Brasil
www.htmicron.com.br

HTLRBL32L-HTCrypto-LoRaBase

Firmware Specification Document

Code: [Palavras-chave]

Doc. Type: Specification Document

Date: 02/12/2021

Revision: PRELIMINARY

Classification: [Comentários]

1. SUMMARY

2. TABLE LIST

DOCUMENT INFO

This document provides information about the specifications of all blocks that will compose the system in package.

REVISION

Version	History	Date	Authors
00	- Initial draft	02/12/2021	Christian R Lehmen
1	Bug fixes and new features	11/03/2022	Christian R Lehmen

APPROVAL

Position	Name	Date	Remarks

3. INTRODUCTION

This application is meant to be a simple and easy to use LoRaWAN showcase and also a template from which you can derive your custom firmware. This document will show you how to test and customize this firmware to your own needs.

The firmware works by basically transmitting a LoRaWAN packet with custom payload periodically. It is based on the LoRaWAN 1.0.2 rb specification. Can operate on AU915, EU868, and US915 regions with class A and works on ABP/OTAA activation modes.

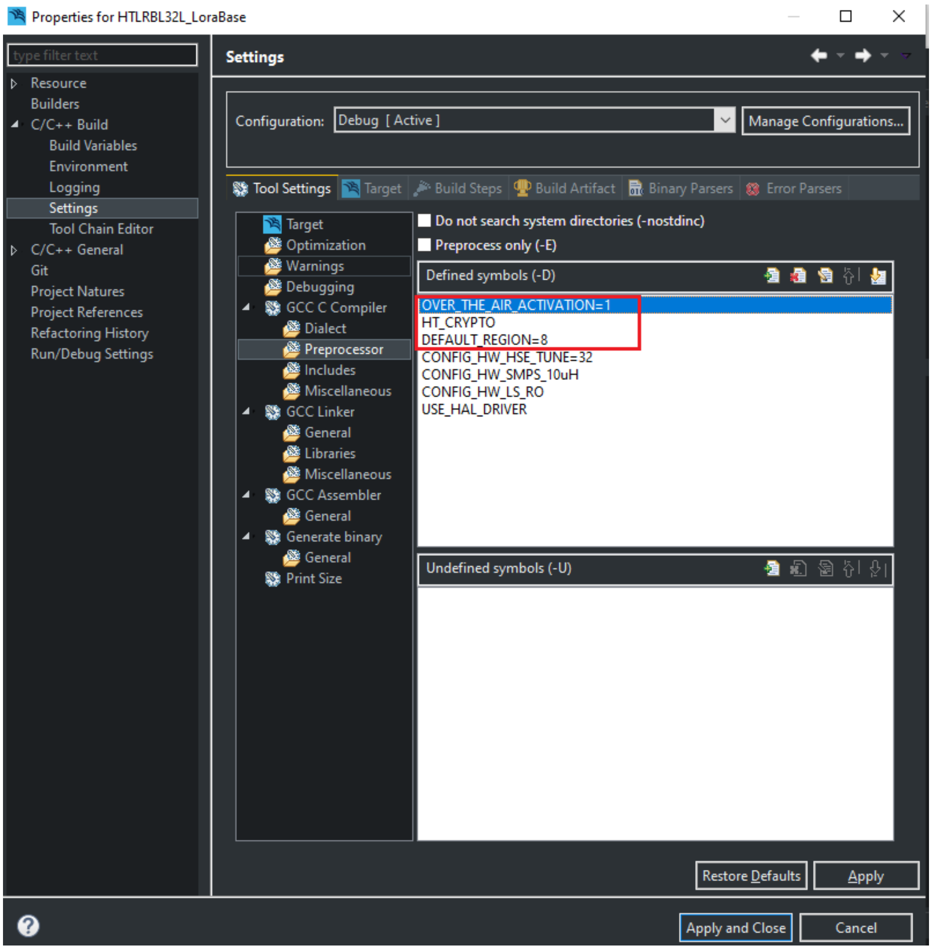
4. DEVICE CONFIGURATION

There are some important parameters that must be configured before you can compile and flash the firmware in the device, such as: LoRaWAN keys, activation type (ABP/OTAA), region of operation, and other configurations that need to match your specific case. This section will go through the most important configs and how you can change them.

4.1. Preprocessor Directives

There are three configuration options that need to be addressed in the preprocessor directives of the compiler: activation mode, region, and HT crypto. They're accessible through Wise-Studio's IDE, as follows:

Right click on the project -> Properties -> C/C++ Build -> Settings -> GCC C Compiler -> Preprocessor



These are the options that must be configured before compiling the code:

- **OVER_THE_AIR_ACTIVATION**: 1 for OTAA (Over the Air Activation) and 0 for ABP (Activation by Personalization)
- **HT_CRYPTO**: Its definition enables/disables the use of the HT Crypto library and secure element (if your hardware version supports it)
- **DEFAULT_REGION**: Region specification based on the document RP002-1.0.2 LoRaWAN® Regional Parameters. The pre-certified available options are on Table 1.

Value	Region
1	AU915
5	EU868
8	US915

TABLE 1 – REGION VALUES

4.2. HT Crypto

If you have the hardware version that has the Hardware Secure Element embedded, you can enable the HT Crypto library within this project. The HT Crypto library is a cryptographic solution that also has an interface with the Hardware Secure Element (HSM), where it stores your LoRaWAN root keys, greatly increasing the security of your device against attacks that aim to clone your device or intercept messages. If you enable the HT Crypto solution, be sure to check the Key Provisioner Firmware Manual before running this firmware. There you will learn how to store your root keys inside the HSM.

4.3. LoRaWAN Keys

For the cases that doesn't use the HT Crypto solution the LoRaWAN root keys must be set directly on code. To do that you must edit the file "Inc/lorawandefines.h", where addresses and keys for each activation mode (ABP or OTAA) are set up.

4.4. Default Transmission Channels

Each region has around 64 different transmission channels available that can be used to send uplinks, but on most cases your gateway only supports 8 of those channels. So, you must configure your device to match those channels. Each region has a specific file on the project that must be edited to configure which channels are going to be used.

Follow the project path and open the file according to your region:

HTLRBL32L-SDK->LoRaWAN->lorawan_specifics->Mac->region->RegionAU915.c
HTLRBL32L-SDK->LoRaWAN->lorawan_specifics->Mac->region->RegionUS915.c

The function **RegionUS915InitDefaults** or **RegionAU915InitDefaults,** depending on your region, has a channel mask, that must be configured accordingly:

```
NvmCtx.ChannelsDefaultMask[0] = 0x00FF;
NvmCtx.ChannelsDefaultMask[1] = 0x0000;
NvmCtx.ChannelsDefaultMask[2] = 0x0000;
NvmCtx.ChannelsDefaultMask[3] = 0x0000;
NvmCtx.ChannelsDefaultMask[4] = 0x0000;
NvmCtx.ChannelsDefaultMask[5] = 0x0000;
```

FIGURE 1: DEFAULT CHANNEL MASK

Each bit of the mask corresponds as an enabled channel and each array index holds 16 channels, on Figure 1 you can see that only channels 0 to 7 are enabled (0x00FF= 0000 0000 **1111 1111**).

The same rule applies for array indexes 1 to 5. FIGURE 2 has enabled channels 24 to 31 (0xFF00= **1111 1111** 0000 0000). while disabled all others.

```
NvmCtx.ChannelsDefaultMask[0] = 0x0000;
NvmCtx.ChannelsDefaultMask[1] = 0xFF00;
NvmCtx.ChannelsDefaultMask[2] = 0x0000;
NvmCtx.ChannelsDefaultMask[3] = 0x0000;
NvmCtx.ChannelsDefaultMask[4] = 0x0000;
NvmCtx.ChannelsDefaultMask[5] = 0x0000;
```

FIGURE 2: CHANNEL CONFIGURATION EXAMPLE

Through this method you can set any combination of channels you require.

The channel number and frequency are as follows:

Channel n°	AU915	US915
0	915200000	902300000
1	915400000	902500000
2	915600000	902700000
3	915800000	902900000
4	916000000	903100000
5	916200000	903300000
6	916400000	903500000
7	916600000	903700000
8	916800000	903900000
9	917000000	904100000
10	917200000	904300000
11	917400000	904500000
12	917600000	904700000
13	917800000	904900000
14	918000000	905100000
15	918200000	905300000
16	918400000	905500000
17	918600000	905700000
18	918800000	905900000
19	919000000	906100000
20	919200000	906300000
21	919400000	906500000
22	919600000	906700000
23	919800000	906900000
24	920000000	907100000
25	920200000	907300000
26	920400000	907500000
27	920600000	907700000
28	920800000	907900000
29	921000000	908100000
30	921200000	908300000
31	921400000	908500000
32	921600000	908700000
33	921800000	908900000
34	922000000	909100000
35	922200000	909300000
36	922400000	909500000
37	922600000	909700000
38	922800000	909900000
39	923000000	910100000
40	923200000	910300000
41	923400000	910500000
42	923600000	910700000
43	923800000	910900000

44	924000000	911100000
45	924200000	911300000
46	924400000	911500000
47	924600000	911700000
48	924800000	911900000
49	925000000	912100000
50	925200000	912300000
51	925400000	912500000
52	925600000	912700000
53	925800000	912900000
54	926000000	913100000
55	926200000	913300000
56	926400000	913500000
57	926600000	913700000
58	926800000	913900000
59	927000000	914100000
60	927200000	914300000
61	927400000	914500000
62	927600000	914700000
63	927800000	914900000
64	915900000	903000000
65	917500000	904600000
66	919100000	906200000
67	920700000	907800000
68	922300000	909400000
69	923900000	911000000
70	925500000	912600000
71	927100000	914200000

TABLE 2 – CHANNEL CONFIGURATION

Channels 65-72 are used by exclusively on DR6(AU915) and DR4(US915).

4.5. Custom Payload Transmission

On this application, an uplink with a generic message is sent every 15 seconds. To send your own uplink message all you need to do is call the function "lorawan_send" with your custom payload. The function is located on the "lorawan_setup.c" file. After you call this function the LoRaWAN state machine is set in motion which will that care of the TX and RX processes.

```
void lorawan_send(lora_AppData_t *appData){

   printf("\n-LoRaWAN TX-\n");

   if ( LORA_JoinStatus () != LORA_SET)
  {
   /*Not joined, try again later*/
   LORA_Join();
   return;
  }
   appData->BuffSize = strlen((const char*)appData->Buff);

  LORA_send( appData, LORAWAN_DEFAULT_CONFIRM_MSG_STATE);

}
```

FIGURE 3: LORAWAN_SEND FUNCTION

4.1. Max Payload

Every region and datarate has its own maximum payload size, you need to keep it in mind when customizing your own payload. Table 3 contains the maximum available payload sizes for each configuration. Note that the maximum value might be a little smaller depending on if some additional configuration bits are added automatically to the packet by the LoRaWAN stack.

Datarate	AU915	EU868	US915
0	51	51	11
1	51	51	53
2	51	51	125
3	115	115	242
4	242	242	242
5	242	242	Invalid
6	242	242	Invalid
7	Invalid	242	Invalid

TABLE 3 – MAXIMUM PAYLOAD

4.2. Serial Terminal Setup

The UART interface can be used by connecting the pins TX(PA9) and RX(PA8) to a USB-Serial converter and connecting to a computer or simply connecting it to another microcontroller with a UART interface.

UART configuration required to connect:

- Baud rate: 115200
- Data bits: 8
- Stop bits: 1
- Parity: none
- Flow Control: none
- Transmitted text: Append LF

4.3. Termite Setup

One of the most widely used software for UART communication using a computer is [Termite](). Figure 4 – Termite setup displays the configuration required to connect to the HTLRBL32L.

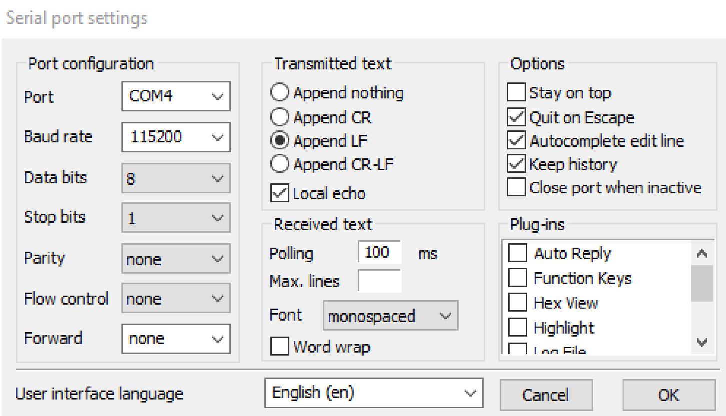


FIGURE 4 – TERMITE SETUP