Introduction

This document describes the software architecture and implementation of the host controller application firmware example included in the STSW-ST8500GH software package.

The package provides the software ecosystem for ST’s G3-PLC Hybrid PLC & RF connectivity technology evaluation, based on the ELVKST8500GH868, EVLKST8500GH915 kits that include all the functions required for plug-and-play communication networking.

The host controller application firmware example for STM32G070RB, based on FreeRTOS, allows testing the PLC and RF communication exploiting the IPv6 layer interface of the ST8500 modem. The G3-PLC Hybrid PLC & RF communication stack has full flexibility to be configured in any of the available bandplans for both PLC (CEN-A, CEN-B or FCC) and RF (according to the RF Sub-GHz module selection).

Messages between two nodes in the PLC & RF hybrid network are sent over the best available medium: PLC or RF. The media selection for each link in the network is done automatically and adjusted dynamically, enabling highly efficient hybrid mesh networking.

The ST hybrid PLC & RF solution is based on open standards and enables seamless integration into existing G3-PLC networks and adoption in multiple applications and systems.
1 G3-PLC Hybrid PLC & RF software solution overview

1.1 What is the G3-PLC Hybrid PLC & RF?

The G3-PLC Hybrid is the first industry hybrid communication standard offering extended capabilities for Smart Grid and IoT applications in one seamlessly managed network over both wired and wireless media.

The hybrid protocol stack is built using open standard IEEE 802.15.4-2015 in addition to the existing G3-PLC protocol. Each device in the mesh network can use PLC as well as RF for communication. Depending on the actual conditions in the field, messages between two devices are sent over the most reliable channel available. The channel selection for each link in the network is done automatically and adjusted dynamically.

In many network conditions, none of the communication technologies can consistently achieve connectivity >99% on their own. The hybrid solution maximizes coverage and connectivity, avoiding the high cost that would be associated to deployment of a different solution to achieve the remaining 1% connectivity. The G3-PLC Hybrid profile can provide a more efficient and cost-effective solution for smart grids, smart cities and industrial applications.

The G3-PLC Hybrid profile is:

- fully compatible and interoperable with existing G3-PLC implementations, so it is possible to mix hybrid and non-hybrid nodes in the same network;
- available for all PLC bandplans and supporting a wide range of non-licensed Sub-GHz bands worldwide.

1.2 Protocol basics

The G3-PLC Hybrid PLC & RF protocol stack (see Figure 1) is based on the well-known G3-PLC protocol standard. To get the best from the two communication technologies involved, Narrow-Band PLC and Sub-GHz RF, the existing G3-PLC MAC layer has been replicated (and adapted) on top of the RF PHY, and a Hybrid Abstraction Layer has been developed on top of the two MAC layers, to guarantee seamless integration of the two media in one single managed network.
The principle of operation is quite simple: each node in a network having hybrid connectivity decides which medium to use to reach better performances and coverage, realizing a hop-by-hop automatic selection which is totally transparent to the end user. At the same time, the high flexibility of the implementation allows to mix hybrid nodes with PLC-only and/or RF-only nodes in a single network.

1.3 Supported Hardware and evaluation boards

The ST G3-PLC Hybrid PLC & RF is based on three main devices (ST8500, STLD1 and S2-LP) building together the G3-PLC Hybrid PLC & RF modem, with an STM32 microcontroller for the customer application firmware.

- The ST8500 programmable power line communication modem system-on-chip is at the heart of the system, implementing the full G3-PLC Hybrid PLC & RF stack down to layer 2 (MAC layer) and the PHY layer of the PLC medium.
- The S2-LP ultra-low power, high performance, Sub-GHz transceiver implements the PHY layer of the RF medium and the Sub-GHz RF radio.
- The STLD1 is the Line Driver for the PLC medium.

The ST G3-PLC Hybrid PLC & RF implementation can run on several evaluation and development kits. Listed below are the ones where the STSW-ST8500GH STM32 firmware described in this document can be used without adaptation. Although the EVALKITST8500-1 can run the same protocol stack in PLC only mode, the STM32 code provided cannot be used without an adaptation.
### Table 1. ST8500 G3-PLC Hybrid PLC & RF supported boards

<table>
<thead>
<tr>
<th>Part number</th>
<th>Hybrid support</th>
<th>PLC connectivity</th>
<th>RF connectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVLKST8500GH868</td>
<td>YES</td>
<td>0-500 kHz</td>
<td>863-870 MHz</td>
</tr>
<tr>
<td>EVLKST8500GH915</td>
<td>YES</td>
<td>0-500 kHz</td>
<td>860-940 MHz</td>
</tr>
</tbody>
</table>
2 The G3-PLC Hybrid PLC & RF software package

The STSW-ST8500GH contains a software package for the STM32G070RB microcontroller based on STM32CubeMX V6.3.0 and STM32CubeIDE V1.8.0. The structure of the package is described hereafter (please refer to the specific release note to check if there are changes in your version).

- NUCLEO-G070RB
  - \Documents
  - \Drivers
  - \G3_Applications
  - \Inc
  - \Middlewares
  - \Modules
  - \Print_Applications
  - \Src
  - \Startup
  - \User_Applications
  - NUCLEO_G070RB.ioc
  - STM32G070RBTX_FLASH.ld

The \Drivers folder contains CMSIS and STM32 HAL (Hardware Abstraction Layer) files for the selected platform (STM32G070RB microcontroller).

The \G3_Applications folder contains all the protocol specific firmware, comprising the API and the G3-PLC applications.

The \Inc and \Src folders contain the source code generated by the STM32CubeMX project and framework code, while the \Middlewares folder contains the FreeRTOS specific source files.

The \Modules folder contains the source code for all the modules used in the application.

The \Startup folder contains the assembly files responsible for the microcontroller preliminary startup operations (stack pointer setup, filling the BSS section with zeros, system initialization, calling the main function…).

The \Print_Applications folder contains the files related to the Print task, responsible for the printing operations to the serial terminal.

The \User_Applications folder contains the files related to an example of UDP user application: a UDP sender/receiver implementation with its FreeRTOS task, serial terminal handling and several utility functions.

The \Documents folder includes a quick start guide on the User application and the release note of the package; you should check this document to see the changes that have occurred since the last official release.

The file NUCLEO_G070RB.ioc is the STM32CubeMX project file. You can use it to modify and/or regenerate the project (to modify the toolchain, the pin assignment, the target STM32 etc.) within the constraints of the STM32CubeMX environment.

The file STM32G070RBTX_FLASH.ld is the linker script file. It lists and describes all the memory section allocated inside the microcontroller (FLASH and RAM).
3 STM32CubeMX project

3.1 Project description
When first opened, the STM32CubeMX project shows, on the left side, the list of categories the settings refer to, while on the right panel the default view is on the STM32 pinout. By moving on the active pins (the ones not grayed), it is possible to see how they are configured and, when clicking, the alternative settings are shown.

Figure 4. STM32CubeMX project initial view

To see/change the configuration of each peripheral, a side panel is displayed by clicking on a specific peripheral from the left side list, with related operating mode and configuration parameters.
Any information about the STM32CubeMX ecosystem can be found on the st.com website [www.st.com](https://www.st.com/content/st_com/en/stm32cube-ecosystem.html).

### 3.2 FreeRTOS subsystem

The FreeRTOS middleware is generated automatically by STM32CubeMX. The overall FW structure is shown in Figure 6.
The STM32CubeMX generates the STM32 HAL/CMSIS drivers and configures the middleware (in this case, FreeRTOS).

The FreeRTOS configuration parameters can be modified in the FreeRTOS tab under the Middleware section inside STM32CubeMX (see Figure 7).

For this project, the firmware package STM32Cube_FW_G0_V1.5.0 has been used as a starting point.
The G3-PLC Hybrid PLC & RF SW project makes use of the CMSIS_V2 interface and generates four tasks:

- The FreeRTOS Default task that blinks an LED to signal it is working and stops in case of fault/stack overflow.
- The FreeRTOS G3-PLC task that handles the PLC communication. A complete description of the \texttt{g3\_task} is given in Section 4.
- The FreeRTOS User task that handles an example of UDP application. A complete description of the \texttt{user\_task} is given in Section 5.
- The FreeRTOS Print task that handles the low-level printing operation to serial terminal. A complete description of the \texttt{print\_task} is given in Section 6.

In addition, one event flag, one stream buffer, four semaphores and one mutex are used by the OS:

- \texttt{eventSync}: event flag used to synchronize all application tasks (User task, G3-PLC task, Print task) after their initialization.
- \texttt{hostStreamBuffer}: stream buffer used to transfer data to print to the Print task.
- \texttt{mutexPrint}: mutex used to make sure that only one task sends data to the Print task at a time.
- \texttt{semHostTxStart}: semaphore used to unblock the Print task whenever a \texttt{UserTerm\_LowLevelPrint} function is called.
- \texttt{semHostTxComplete}: semaphore used to keep the Print task blocked while the data is being transmitted to the serial terminal.
- \texttt{semConfirmation}: semaphore used to make sure that no new request is sent to the ST8500 before a confirmation to the previous one is received.
- \texttt{semSPI}: semaphore used to block the User task during the SPI transaction with the SPI Flash memory.

### 3.3 UART interface modules

The G3-PLC Hybrid PLC & RF SW project includes several modules related to UART communication between the ST8500, the STM32 and the PC:

- ST8500 Host UART Interface
- User UART Interface
- ST8500 Host UART Pass-through
This chapter describes the implementation of these modules in detail.

### 3.3.1 ST8500 Host UART Interface

The ST8500 Host UART Interface module enables the control of the ST8500 Host UART Interface, that is the UART interface of the ST8500 implementing commands related to the G3-PLC hybrid protocol stack. The ST8500 Host UART Interface module basically takes care of the low-level UART messaging between the STM32 and the ST8500; it is implemented in the file `host_if.c`.

Besides the initialization and service functions, the core of the operations is managed by three functions:

- **g3_hif_uart_rx_handler**, called inside the ST8500 Host UART ISR when the reception is complete (`HAL_UART_RxCpltCallback` inside `callbacks.c`), receives the message coming from the ST8500 Host Interface and then queues it for the G3-PLC task when the UART reception has ended and the message length has been verified;
- **g3_hif_uart_send_message**, called from the G3-PLC task main loop, which initiates the transmission of a message to the ST8500 Host Interface;
- **g3_hif_uart_tx_handler**, called inside the ST8500 Host UART ISR when the transmission is complete (`HAL_UART_TxCpltCallback` inside `callbacks.c`), resets a variable to allow the transmission of a new message to the ST8500 Host Interface.

### 3.3.2 User UART Interface

The User UART Interface module controls the UART dedicated to the interaction between the device connected through a USB cable (a PC is assumed, for instance) and the STM32 User task. The User Interface UART is connected to the PC through the ST-LINK USB on the NUCLEO-G070RB board. This module, implemented in the file `user_if.c`, receives the serial input from the PC directly when the UART reception ISR is called, while the output is sent to a dedicated FreeRTOS low priority task, that is the Print task.

Besides the initialization and service functions, the core of the operations is managed by three functions:

- **UserTerm_UartIsrRx**, called inside the User UART ISR when the reception is complete (`HAL_UART_RxCpltCallback` inside `callbacks.c`), requests data from the User UART Interface to put it in a dedicated buffer (`TermRxFifo`);
- **UserTerm_GetCommand**, called inside the User task, extracts the data in the dedicated buffer to make it available for the operations of the task;
- **UserTerm_LowLevelPrint**, transmits formatted data to a stream buffer (`hostStreamBuffer`), when FreeRTOS is running. The mutex `mutexPrint` is used to make sure that only one task at a time sends data to the stream buffer. If FreeRTOS is not running, it transmits data directly to the UART;
- **UserTerm_UartIsrTx**, called inside the User UART ISR when the transmission is complete (`HAL_UART_TxCpltCallback` inside `callbacks.c`), releases a semaphore to allow the transmission of new data through the User UART Interface.

The Print task takes care of extracting the data from the stream buffer and transmitting it to the User UART (see Section 6).

### 3.3.3 ST8500 Host UART Pass-through

The ST8500 Host UART Pass-through module gives direct access to the ST8500 Host Interface from the PC via the NUCLEO-G070RB USB connector. When activated through a specific switch on the board, this module takes complete control of the STM32, reconfiguring the pins of the two UARTs (the one connected to the ST8500 and the one connected to the USB) as GPIOs, to provide a transparent link. The serial communication is forwarded from the USB to the ST8500 by the STM32 with an EXTI (External Interrupt) mechanism that controls the output GPIOs (former UART TX pins) whenever the input GPIOs change (former UART RX pins). In other words, the ST8500 Host UART Pass-through module replicates the signals coming from the User UART on the ST8500 Host UART and vice versa, to allow the user to communicate directly with the ST8500, from the USB port (that carries the User UART signals).

The ST8500 Host UART Pass-through module is activated with the `activatePassthrough` function inside the `passthrough.c` file which, in order:

1. De-initializes all connectivity peripherals.
2. Disables the System Tick interrupt (`TIM6_IRQHandler`).
3. Re-configures the RX pins of the ST8500 Host UART and User UART as EXTI.
4. Sets high the TX pins of the ST8500 Host UART and User UART.
5. Re-configures the RX pins of the ST8500 Host UART and User UART as outputs.
6. Enables the interrupt for the EXTI RX pins.
7. Turns ON an LED to indicate that the pass-through is active.
8. Enters an infinite “Wait-For-Interrupt” cycle.

The Pass-through activity is indeed performed using interrupts. In the file `stm32g0xx_it.c` the `EXTI2_3_IRQHandler` and the `EXTI4_15_IRQHandler` functions perform the minimal operations required to perform the Pass-through, that consist of setting the TX pin of a UART to the same level of the RX pin of the other UART, whenever it changes.

It must be noted that, to support high baud rates, the IRQ handlers must execute the smallest possible number of instructions. For this reason, the use of HAL functions is not recommended inside the IRQ handler functions.

### 3.4 Task communication mechanism

Task communication between the G3-PLC task and the User task is achieved by two queues named `user_to_g3_queue` and `g3_to_user_queue`. In addition, the reception from the ST8500 Host Interface utilizes a queue named `hif_to_g3_queue` to transfer the messages from the UART reception ISR handler to the G3-PLC task.

These are monodirectional queues, and the usage is as follows:

- **user_to_g3_queue** is used by the User task to queue commands to be sent to the G3-PLC task and by the G3-PLC task for self-generated commands to be sent. The access functions are:
  - `taskCommPut_user_g3`, to queue a message
  - `taskCommGet_user_g3`, to de-queue a message
  - `taskCommAvailable_user_g3`, to check if a message is available in the queue

- **g3_to_user_queue** is used by the G3-PLC task to send messages to the User task. The access functions are:
  - `taskCommPut_g3_user`, to queue a message
  - `taskCommGet_g3_user`, to de-queue a message
  - `taskCommAvailable_g3_user`, to check if a message is available in the queue

- **hif_to_g3_queue** is used by the ST8500 Host Interface to transfer the messages received from the ST8500 to the G3-PLC task. The access functions are:
  - `taskCommPut_hif_g3`, to queue a message
  - `taskCommGet_hif_g3`, to de-queue a message
  - `taskCommAvailable_hif_g3`, to check if a message is available in the queue

These functions rely on the CMSIS_V2 queuing mechanism through the functions:

- `osMessageQueueNew`
- `osMessageQueuePut`
- `osMessageQueueGet`
- `osMessageQueueGetCount`

The message format is defined by the `task_comm_msg_t` structure: the `cmd_id` is the command code valid for the ST8500 Host Interface; `buff` is the pointer to the data associated with the command; `size` is the length of the data pointed by `buff` in bytes.

To avoid deep copy operations in the ISR, which could lead to a loss of data at higher baud rates, or with low speed STM32 devices, shared buffers are used to exchange data between the UART and the G3-PLC task; these buffers are dynamically allocated through a specific module (`mem_pool.c` and `mem_pool.h`). Eight 1600-byte shared buffers are defined; to allocate and deallocate the buffers the following interface functions are defined (see file `mem_pool.h`):

- `Mem_Alloc`, to allocate a buffer
- `Mem_Free`, to deallocate the buffer
- `Mem_FreeAll`, to deallocate all the buffers and free the entire memory pool.
3.5 ST8500 image download

When activated through a specific switch on the board, this module downloads both the RTE and the PE images from the STM32 SPI Flash memory to the ST8500, during the startup, before initializing the operative system (FreeRTOS). This is handled entirely by the `downloadImageToST8500` function, that, in order:

1. Saves the current baud rate of the Host UART.
2. Validates the ST8500 images in the STM32 SPI Flash, if they still need to be validated.
3. Changes the baud rate of the Host UART to 9600.
4. Searches for the RTE and PE images, looking for primary ones.
5. If at least one of the two images is missing, searches for the missing images, looking for secondary ones.
6. Sets the communication baud rate (and the Host UART baud rate) to 921600.
7. Downloads the RTE image to the ST8500.
8. Downloads the PE image to the ST8500.
9. Restores the original baud rate of the Host UART.

To download the images from the STM32 SPI Flash, it is necessary to store them on the SPI Flash in the first place. This can be done with STM32CubeProgrammer, following the procedure described in Section 8.
4 G3-PLC applications

4.1 Introduction
The G3-PLC task implements three modules:

- G3-PLC Config module
- G3-PLC Boot module
- G3-PLC Keep-alive module

This section describes the G3-PLC task and gives some hints on how it could be adapted to fit specific cases.

4.2 The G3-PLC Config module
The G3-PLC Hybrid PLC & RF node can be configured on several parameters: the role of the node in the network (PAN Device or PAN Coordinator), the PLC band (CENELEC A, CENELEC B, FCC), the RF frequency and transmission mode, and so on. Such parameters must be configured at the startup of the node to ensure that it properly behaves in the network.

In case an SPI Flash memory is connected to the ST8500, all the configurations are written in this Non-Volatile Memory (NVM) at specific sectors; the NVM content is retrieved and the ST8500 G3-PLC configuration restored at each power-on/HW reset. The NVM management is almost transparent to the user but, to avoid conflicts between the ST8500 retrieving parameters from the NVM and the Host trying to configure the modem, after each power-on/HW reset event, the host waits until the ST8500 notifies the end of NVM reconfiguration by sending the message HOSTIF-HWRESET.Confirm on the Host Interface.

The G3-PLC Config module takes care of customizing the configuration of the node in three steps:

1. G3-PLC attributes table initialization
2. G3-PLC platform configuration
3. G3-PLC attributes configuration

4.2.1 Attributes table initialization
The list of parameters to be configured is stored into a static array of type G3_LIB_PIB_t; the attributes are inserted in the attributes table with the function g3_attrib_tbl_add_elem, defined in the file g3_app_attrib_tbl.c, that is called multiple times inside g3_fill_attribute_tbl, that is executed during the initialization. It is possible to add attributes only for a specific role (only for coordinator or only for device) or for both coordinator and device.

In case the node is configured as a PAN coordinator, the only two mandatory parameters to be configured are the MAC_PAN_ID and the MAC_SHORTADDRESS_ID.

In case the node is configured as a PAN device, the only mandatory attribute to be set is ADP_EAPPSKKEY_ID (as the Short Address is configured during the Bootstrap procedure).

The user can easily extend this configuration to adapt to their needs, by adding rows to the two tables with the g3_attrib_tbl_add_elem function (the MAX_ATTRIBUTE_NUMBER macro must be adjusted to the maximum number of attributes inserted). During the node startup, this table is prepared to be used by the G3-PLC attribute configuration described below.

4.2.2 G3-PLC platform configuration
The G3-PLC Hybrid PLC & RF stack must be properly initialized and configured before starting the operations. The G3-PLC Config module performs this operation at the startup of the G3-PLC task, in the g3_app_conf_start function, by sending an SW reset request to the ST8500.

The hi_g3lib_swresetreq_set function inside of g3_app_conf_start accepts three parameters:

- The PLC operating band (Cenelec A, Cenelec B or FCC)
- The Device type (PAN Coordinator or PAN Device)
- The Operating mode (HI_PHY_MODE, HI_MAC_MODE, HI_ADAP_MODE, HI_BOOT_MODE, HI_IPV6_ADAP_MODE, HI_IPV6_BOOT_MODE)
The G3-PLC task supports all the PLC operating bands and Device type, but it requires the modem to be set in HI_IPV6_BOOT_MODE to properly operate.

Upon reception of the G3LIB-SWRESET.Confirm message on the ST8500 Host Interface, a DBG-TOOL request is sent to obtain the ST8500 firmware version and the status of the RF module. In case the firmware version supports the RF functionality (major version greater or equal to 5) and the RF module is detected (RIconf field equal to “1”), the configuration of the RF module is performed by the G3-PLC task by calling the g3_conf_fsm_received_dbgtool_cnf function. It simply feeds the structure containing the RF config parameters using hard-coded values, depending on the selected RF module (X-NUCLEO-S2868A2 or EVALS2915A1/A2).

Note that the frequency range for the RF link is fixed by the selected configuration (both coordinator and device have 863.1 MHz option and 915 MHz option), so, to ensure that all nodes of the network can communicate via RF, it might be necessary to set a custom RF frequency for all nodes. It is possible to set a custom frequency by editing the value of the DEFAULT_RF_FREQ macro.

Please check the compatible frequency range for EVLKST8500GH868 and EVLKST8500GH915 in the respective documentation on www.st.com.

4.3 G3-PLC attributes configuration

Once the RF parameters are configured, the G3-PLC Config module handles coordinator and device differently. In case of PAN device, the attributes are set from the G3-PLC task right after the RF parameters are configured. In case of PAN coordinator, the task waits for the default server start confirmation and then, inside g3_conf_fsm_received_default_srvstart_cnf, stops the server.

After the server stop confirmation has been received, in g3_conf_fsm_received_srvstop_cnf, the G3-PLC Config module starts setting all the attributes found in the g3_app_attrib_tbl table by calling g3_conf_set_attr multiple times (only one attribute is set at a time). Once all attributes are set, a server start request is sent, using the PAN ID and the short address from the attributes (the short address must be 0).

4.4 The G3-PLC Boot module

The G3-PLC Boot module makes use of the G3-PLC Hybrid PLC & RF bootstrap implementation to manage the registration of each PAN Device in the network to the PAN Coordinator. It is important to remember here that it is required to run a network having (only) one PAN Coordinator and (at least) one PAN Device.

4.4.1 PAN device implementation

In G3-PLC Hybrid PLC & RF networks, at startup each PAN Device tries to locate a reachable PAN through the Discovery procedure. The result of this procedure is that a certain number of Agents are identified (the PAN Coordinator and/or some PAN Devices already connected to the network). The list of agents is then passed to the host application through the Boot-Device-Pansort.Ind message from the ST8500 Host Interface.

Since the PAN Device must decide which agent shall be used to join the network, it is up to the host application to sort the list of agents by route cost. The function g3_app_pansort_handle_pansort_ind (inside g3_app_pansort.c) sorts the list of agents using the route cost to PAN Coordinator as metric and, at the end, sends back the ordered list of agents to the ST8500 Host Interface through the message Boot-Device-Pansort-Req.

At this point, the G3-PLC Hybrid PLC & RF implementation starts trying to join the network through the first agent and, in case of failure, scans the entire table until the node is registered.

4.4.2 PAN coordinator implementation

Each time a PAN Device asks to enter the network, independently from the fact that the selected Agent is the PAN Coordinator or another PAN Device, the request is eventually forwarded to the ST8500 acting as PAN Coordinator, which generates a G3BOOT-SRV-GETPSK.Ind message on the ST8500 Host Interface to the G3-PLC Boot module. The G3-PLC Boot module gets from it the Pre-Shared Key (PSK) of the specific PAN Device and the short address the PAN Coordinator wants to assign to it.

The PAN Coordinator assigns the short address to the requesting PAN Devices either with static or dynamic allocation, using the associated PSK. The static allocation is supported by g3_boot_main_table table (inside g3_app_boot.c), which is hard-coded in the PAN Coordinator G3-PLC Boot module: when a PAN Device whose Extended Address is listed in the table it tries to join the network, then the corresponding Short Address and PSK are assigned.
If the Extended Address of the PAN Device is not present in the table, then it is assigned a progressive Short Address and a default PSK, by inserting the device in the `g3_boot_temp_table` table.

In a real implementation, the dynamic allocation should be avoided, so that a malicious node is not allowed to access the network. Similarly, a hard-coded table is not the right choice to store this information and, for this purpose, the use of a secure element to store locally the table, or the use of a secure connection towards an authentication server, is recommended.

The access to the network is granted in the `g3_app_boot_handle_srv_getpsk_ind` function.

As a general hint, the user could remove all the instructions inside the first `if` branch of the function, leaving only a `return` statement, to avoid the automatic registration of unknown nodes.

### 4.5 The G3-PLC Keep-alive module

In a real network, the channel conditions can vary during uptime, and sometimes it may happen that a PAN Device cannot exchange data with the PAN Coordinator (or vice versa), meaning that its route to the PAN Coordinator is not valid anymore. Even if the channel conditions remain acceptable, the entries in the routing table of the nodes have a limited validity time; if there is no communication for a longer time, the entry is invalidated.

Usually this condition is asserted as soon as the PAN Device tries to send data to the PAN Coordinator (or vice versa); this triggers a Route Repair mechanism which takes time to be completed, significantly increasing the round-trip-delay of the triggering data message and the overall traffic in the network. In many cases, this is not a problem, but there are cases where this delay may become unacceptable, either because it is larger than the validity time of the data transferred, or because it may lead to a temporary network instability if many nodes try to recover their route to the PAN Coordinator in a short period of time.

To lower the occurrence of this event, especially on networks where PAN Devices do not communicate regularly with the PAN Coordinator, it is good practice to implement a Keep-alive mechanism, which should be able to:

- Periodically monitor the connection
- Try to repair the route if the connection is not stable (this is done automatically by the G3-PLC Hybrid PLC & RF implementation)
- Disconnect from the network and restart the bootstrap procedure in case the failure cannot be recovered.

The Keep-alive module takes care of this process.

#### 4.5.1 PAN device implementation

The Keep-alive module in the PAN Device keeps track of the last time a data communication with the PAN Coordinator has been successfully performed. If there is no communication for more than `KA_DEVICE_LEAVE_TIME`, the device leaves the PAN and it retries the bootstrap procedure; this is managed by the `g3_ka_dev_check_timeout` function, which sends a device leave request when called. The Keep-alive timeout for the device is checked every `KEEP_ALIVE_CHECK_PERIOD` milliseconds.

When the PAN Device leaves the PAN, it tries to reconnect to it after `KADEVICE_RECONN_TIME` milliseconds.

#### 4.5.2 PAN coordinator implementation

The Keep-alive module in the PAN Coordinator makes use of the ICMP echo feature to communicate periodically with each PAN Device; the interval between two echo requests (`KEEP_ALIVE_CHECK_NEXT_DELAY`) is configurable.

When a server join indication is received, the new PAN Device is added to the Keep-alive table of the PAN Coordinator. This table contains the short addresses and extended addresses of all the PAN Devices that joined the PAN.

The Keep-alive mechanism uses a “lives” system, that consists of subtracting one “life” each time a ping fails and resetting the lives back to the maximum amount each time a ping is successful. Each device starts with `KEEP_ALIVE_LIVES_N` lives when it joins the PAN.
Every `KEEP_ALIVE_CHECK_PERIOD` milliseconds (60 seconds by default), a Keep-alive event takes place (`g3_ka_panc_check_device` is called), and it lasts `KEEP_ALIVE_CHECK_NEXT_DELAY` milliseconds (1 second by default) multiplied by the number of devices that joined the PAN minus one (if there is only one device, the Keep-alive event is instantaneous). During the “keep-alive” event all devices in the PAN are pinged. The number of lives of each device decreases if that device does not answer, or gets restored to the initial value if it answers. If the number of lives goes to 0, the device gets kicked out of the network. This is managed by the `g3_ka_fsm_check_next_device` function that can call either the `g3_ka_panc_ping_device`, that sends the ICMP echo request to the device, or the `g3_ka_panc_kick_device`, which removes the disconnected device from the Keep-alive table and sends the kick request.

### 4.6 The G3-PLC task

The G3-PLC task is composed of an initialization function, `g3_task_init`, that is executed once after FreeRTOS has started, and a main routine function, `g3_task_exec`, that contains an endless `for` loop.

The `g3_task_init` is responsible the following initialization:

- Timed events utility;
- G3-PLC Boot module;
- ST8500 Host UART;
- G3-PLC Config module;
- G3-PLC Keep-Alive module.

Moreover, `g3_task_init` also starts the ST8500 Host UART reception with interrupts.

The main routine functions, `g3_task_exec`, instead, is composed of two parts:

- a startup block that is executed only once, responsible for receiving the HW reset confirmation and starting the G3-PLC Config module;
- an endless `for` loop that takes care of receiving and transmitting G3 messages, executing the timed events and running the Boot, Config and Keep-Alive modules.

The implemented reception/transmission mechanism is made that only one request can be sent to the ST8500 at a time. Before sending another request, the confirmation to the previous request must be received. To avoid critical blocks, in case the confirmation is missed by error, a timeout for the confirmation reception is present. Each iteration is ended by an `osDelay` call (of only one millisecond) that is necessary to concede computation time to other lower-priority tasks (in this way, the G3-PLC task is forced to go to the `DELAYED` state).
5 User applications

5.1 Introduction
A specific task, named User task, has been defined to handle the user application. The corresponding files are in the User_Applications folder. The User task implements an example of a UDP application launched via a serial terminal.

This section describes the User task and explains how it is handled.

5.2 User task interface with the G3-PLC task
When the G3-PLC task receives messages from the ST8500, these are forwarded to the User task, if needed, via the g3_to_user_queue. The messages from the G3-PLC task to the User task have the g3_user_msg_t format. The cmd_id, buff and size are the same as the ones explained in Section 3.4.

5.3 User task interface with the serial terminal
The User task interacts with the user via any PC serial terminal (like Teraterm) using the User UART Interface port (i.e. STM32G070RB USART2 as defined in STM32CubeMX). The terminal data is handled inside the user_if.c file.

5.3.1 Reception (from terminal to User task)
Bytes received from the UART are first packetized to create a command message. Its content is defined by the user_term_cmd_body_t structure. The TERM_CMD_PAYLD_MAX_SIZE macro sets the maximum length for one string to be sent from the terminal (set to 128). To be considered as valid, a string must end with a CR or LF character.

Once created, the received command message is buffered so that it can be processed by the User task. The whole reception control is defined by the user_term_ctrl_t structure.

Command message packetization and RX buffer feeding are handled under UART ISR.

5.3.2 Transmission (from User task to terminal)
The User task can send data to the terminal by either using the UserTerm_Printf, UserTerm_PrintfWithLabel or UserTerm_PrintfWithNoTs functions, depending on the desired print format. All these functions call the UserTerm_LowLevelPrint function, that sends character strings to the print task, through the host stream buffer.

The stream buffer size is set by the TERM_TX_FIFO_SIZE macro (1024 bytes by default).

The bytes put in the host stream buffer are extracted and sent to the User UART by the Print task, during its execution time. See Section 6 for further details regarding the Print task implementation.

5.4 User task internals
The User task is composed by an initialization function, user_app_init, and a looped routine function, user_app_task. Between the calls of these two functions, each task sets a group event and waits for the other two, that shall be set by the other two tasks. This is useful to sync all tasks and make sure that each one executes its loop function after all initializations have been completed. The user_app_task function consists in handling the messages forwarded from the G3-PLC task, managing the UserG3 FSM (see Section 5.4.1) and the User Interface FSM (see Section 5.4.2). Each iteration is ended by an osDelay call (of only one millisecond) that is necessary to concede computation time to other lower-priority tasks (in this way, the User task is forced to go to the DELAYED state).

5.4.1 G3-PLC task communication management
The User task interface with the G3-PLC task is implemented in user_g3_common.c. This interface management ensures:
• The parsing of incoming messages from G3-PLC task, extracted from the G3-User queue with `taskCommGet_g3_user`, using the `UserG3_MsgHandler` function, that generates user events and calls a specific sub-function, depending on the received message;

• The implementation of UserG3 FSM, the `UserG3_FsmManager` function, that is useful to handle the UDP layer, including the connection setup, the transmission of UDP packets (UDP data requests), the reception of UDP data confirmations from the ST8500 and the reception of UDP packets (UDP data indications). The FSM, at startup, sets all the connections pointed inside the `ConnList` array. The user can add more connections, at his discretion. A packet can be sent through a specific connection by calling the `UserG3_SendUdpData` function using the proper connection number as first parameter.

The implementation of a structure named `user_g3_common_data_t` gathers data to be made available to the User interface FSM (User or G3-PLC events, upcoming UDP transfer content).

5.4.2 User Interface FSM

The User Interface FSM is implemented in `user_if_fsm.c`, with the `UserIf_Fsm_FsmBody` function. Its main goal is to manage the serial terminal display and various actions in the system (e.g. interactions with the G3-PLC layer), depending on terminal user inputs on one side and events coming from the G3-PLC task and the ST8500 Host Interface on the other side. Each time `UserIf_Fsm_FsmBody` is called:

1. User events are evaluated by calling `UserIf_Fsm_ParseUserEvents`.
2. It is checked if the Esc key was typed on the terminal by the user to reset the FSM.
3. The current state function is executed.

The User Interface FSM has two sub-states for each state, one for printing information and instructions on the terminal, and one to acquire input from the user. Some states or sub-states (like the `UserIf_Fsm_UdpTestSingleDataReqState` function) can be organized in multiple steps. By interacting with the terminal, it is possible to navigate through the various states, sub-states and steps, with the possibility to go back to the main menu by pressing the Esc key. A user input command/string is acquired from the User UART Interface buffer by calling the `UserTerm_GetCommand` function.

The actual implementation of the User Interface FSM ensures:

• Exchange of UDP messages for testing purposes;
• Modification to the UDP test connection (destination IPv6 address, remote IPv6 address, remote UDP port, local UDP port);
• Erasure of the STM32 SPI Flash memory (single sector or mass erase);
• UDP transfer of an ST8500 image inside the STM32 SPI Flash memory to another PAN device;
• Sending custom, raw command requests to the ST8500 (allowing low-level control over the platform).

See Section 5.6 for further details about the implementation of the various features.

The implementation of the User Interface FSM is fully upgradable so that it can be easily adapted to further user needs (e.g. other user-defined tests, specific routines for demo purposes).

5.5 Use of serial terminal

Using a PC serial terminal allows the user to start the evaluation of some features like UDP data transfers via some launchable tests/demo. It also displays upcoming specific events such as G3-PLC network bootstrap messages. A serial terminal software must be available on the PC side and a USB cable is connected between the EVALKST8500GH board (USB connector of the NUCLEO board) and the PC.

A typical evaluation setup with two evaluation kits is represented in the following image:
5.6 UDP application example

One goal of the User task is to give an example of a UDP application handled via a serial terminal. The implemented UDP application offers the following features:

- multiple ways to test the exchange of UDP messages (UDP tests, explained in Section 5.6.1);
- reconfigurable UDP connection for UDP tests (UDP Connection Setup, explained in Section 5.6.2);
- ST8500 image transfer from/to the STM32 SPI Flash memory (UDP Image Transfer, explained in Section 5.6.3);
- STM32 SPI Flash memory management (SPI Flash Management, explained in Section 5.6.4);
- Custom ST8500 command forwarding (Custom Commands, explained in Section 5.6.5).

All these features are available in the main menu, that is displayed once the device has connected to the PAN. Each feature can be selected through a specific menu option. Each of the following sub-sections describes one of the features in details.

5.6.1 UDP tests

The UDP tests consist of exchanging UDP messages between two connected devices (coordinator-device or device-device) and are implemented with the \texttt{UserIf\_Fsm\_Udp\_Test\_Xxx} functions called inside \texttt{UserIf\_Fsm\_Fsm\_Body} when the relative menu entry is selected. There are three possible types of UDP tests:

- Basic: a UDP string packet is sent from a device to another;
- Multiple Basic: many UDP string packets are sent from a device to another;
- Loopback: a UDP string packet is sent from a device to another, then it is sent back to the first device, as it is.

The test type is selected when the \texttt{UserIf\_Fsm\_Udp\_Tests\_State} function is called. Once the type of UDP test is selected, one of the \texttt{UserIf\_Fsm\_Udp\_Test\_Basic\_State}, \texttt{UserIf\_Fsm\_Udp\_Test\_Multiple\_Basic\_State} or \texttt{UserIf\_Fsm\_Udp\_Test\_Loopback\_State} functions is called. Each of these functions is divided into multiple guided steps, that depend on the chosen role for the device. A device can indeed act as an originator, or as a recipient/looper, depending on the type of test. For each test, two devices are always required: one must act as an originator, and the other must act as a recipient/looper. In addition, other settings requirements must be met:

- the Remote UDP port of the recipient/looper must match the Local UDP port of the originator, or it must be zero;
- the Local UDP port of the recipient/looper must match the Remote UDP port of the originator, and it must not be zero;
- the Remote IPv6 address of the recipient/looper must match the IPv6 address of the originator, or it must be null;
- the Destination IPv6 address of the originator must match the IPv6 address of the recipient/looper.

In the Loopback UDP test, the destination IPv6 address and UDP port are taken from the received packet header.
Each test function calls UserG3_SendUdpData with TEST_CONN_ID as connection ID, to send the UDP packets, and checks for the UDP data indication event to detect the reception of UDP packets.

5.6.2 UDP connection setup

In order to execute the UDP tests with different Local/Remote UDP port values or a different Destination/Remote IPv6 destination, a menu entry lets the user change the test connection settings. Some dedicated functions named UserIf_Fsm_UdpSetupXxx are called in sequence to acquire from the user the desired connection settings, stored inside a setup_info_t structure. Once all connection parameters are set inside the structure, the UserG3_ModifyUdpConnection function is called to replace the test connection parameters with the ones just inserted. Afterward, the User Interface FSM goes into the UserIf_Fsm_GetConnectionSetupConfirm state, where it prepares and sends the new connection setup to the ST8500 and then waits for its confirmation. For more information about the IPv6 operations, please refer to the AN5603 - ST G3-PLC Hybrid solution.

5.6.3 UDP image transfer

In the EVLKST8500GHxxx kit, the STM32 microcontroller is connected via SPI to an external SPI Flash memory. This is useful to store ST8500 images (RTE and PE type) that can be fed to the ST8500 at startup, if the “Boot from UART” mode is selected with the switches. The UDP application is able to transfer ST8500 images stored in the SPI Flash of a device to other devices connected to the same PAN. Note that each device must have the IPv6 address of the other one as destination. For the UDP transfer, a specific reserved UDP connection is used (with local and remote UDP port equal to 2000 and null remote IPv6). Inside the UserIf_Fsm_UdpTransferState function, the user can select if the transfer is in reception or transmission. Depending on the user selection, either UserIf_Fsm_UdpTransferOriginatorExec or UserIf_Fsm_UdpTransferRecipientExec is called. These functions have several steps necessary to handle the UDP transfer.

The image sender device proceeds with the following steps:
1. Displays the SPI Flash content on the terminal by calling checkMemoryContent.
2. Prompts the user for the selection of the image to send.
3. Once the image is selected, calculates the CRC16 of the image, displays the selected image info again and prompts the user for confirmation.
4. If the user confirms, sends a UDP packet with information regarding the image characteristics to the image receiver.
5. Waits for the confirmation of the UDP packet transmission.
6. Waits for the acknowledging UDP packet from the image receiver.
7. If there is still data to send, reads the SPI Flash to get the next block of image data. Otherwise, ends the procedure.
8. Sends the block of image data as UDP packet to the image receiver.
9. Repeats from step 5.

The image receiver device proceeds with the following steps:
1. Displays the SPI Flash content on the terminal by calling checkMemoryContent.
2. Prompts the user for the selection of the slot for the image to receive.
3. Once the slot is selected, waits for the UDP packet with information regarding the incoming image characteristics.
4. When the UDP packet with the image information is received, displays the incoming image characteristics, and prompts the user for confirmation.
5. If the user confirms, erases the SPI Flash memory slot for the incoming image.
6. Sends an acknowledging UDP packet to the image sender.
7. Waits for the confirmation of the UDP packet transmission.
8. If there is still data to receive, waits for the UDP packet containing the next block of image data from the image sender. Otherwise, ends the procedure.
9. Writes the block of image data inside the SPI Flash.

Both sender and receiver use a user_fsm_transfer_t structure to handle the various procedure steps and keep track of the transfer progress. Note that the SPI Flash is driven inside the User task, that is blocked whenever waiting for the SPI transactions to complete. For this reason, some delays in the terminal prints can be experienced when the SPI Flash is being used.
5.6.4 SPI Flash management

It is possible to examine and, eventually, erase the content of the SPI Flash memory connected to the STM32 microcontroller via SPI. This is entirely handled inside the `UserIf_Fsm_EraseSflash` state function that shows the memory content with the `checkMemoryContent` function, prompts the user to select a specific sector or the whole Flash memory and then proceeds to perform the erasure by either calling `eraseMemorySector`, for a single sector erase, or `eraseMemory`, for a bulk erase.

5.6.5 Custom commands

If the user wishes to send custom requests to the ST8500 to perform operations that the UDP application is unable to perform, it is possible to send low-level raw messages directly to the ST8500. The `UserIf_Fsm_SendRawCommand` management this feature entirely, by requesting a hex string from the user, validating it and then sending it to the ST8500. The valid raw command is indeed put in the User-G3 queue by calling `taskCommPut_user_g3`, and the G3-PLC tasks then proceed to send it to the ST8550. Note that the `UserIf_Fsm_SendRawCommand` function just checks if the inserted string only contains hexadecimal characters (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, a, b, c, d, e, f), while no check is performed regarding the message command ID, the message payload content or the message length. Because of this, it is necessary to make sure that the inserted raw command has a valid length and content. Consult the ST-G3-PLC_RF_Solutions_AN document for more information about ST8500 message structures.

Note: it is not necessary to add the sync field or the CRC16 manually inside the raw command, because those are added at a lower level, inside the `g3_hif_uart_send_message` function of the G3-PLC task.
6 Print applications

6.1 Introduction
A specific task, named Print task, has been defined to handle the printing operation to the serial terminal (User UART). Corresponding files are in the 'Print_Applications' folder. The Print task has a lower priority, in order not to execute when more critical tasks are running, and uses the DMA (Direct Memory Access) to send the terminal data through the User UART. This chapter describes the Print task and explains how it is handled.

6.2 Print task implementation
The Print task, when unblocked, extracts the data from the stream buffer ('hostStreamBuffer'), transmits it to the User UART with the DMA and then remains blocked until all data is transmitted. The Print task repeats these instructions until there is no more data to print.

The task is regulated by two FreeRTOS semaphores:
- `semHostTxStart`, used to block the Print task whenever no data to print is present in the stream buffer. It is released at the end of the `UserTerm_LowLevelPrint` function when data to print is available;
- `semHostTxComplete`, used to keep the task blocked while the DMA is transmitting the data.
7 Extending the G3-PLC Hybrid PLC & RF software example

In this section, a few hints on how the user can customize the code are presented, to add new features or modify/remove the existing ones. As a general remark, to maintain compatibility with the STM32CubeMX environment, and allow automatic regeneration of the source code, the user should add to or modify system generated files (the ones included in the folders Drivers, Inc, Src and Middlewares) only inside a specific section indicated by a couple of markers like the ones reported below, which are present in all the modifiable system generated functions:

```c
/* USER CODE BEGIN <func_name> */
...
/* USER CODE END <func_name> */
```

All the user code typed inside such user code space is kept during regeneration, while the code typed outside is removed and completely lost. Of course, this is not required for the files contained in the G3_Applications folder, the Modules folder, the Print_Applications folder, the User_Applications folder and any source file added by the user.

### 7.1 Extending G3-PLC task with additional features

To extend the G3-PLC task with an additional module, the user should make sure to define:

- An initialization function (if needed) for the new module to be called from the g3_task_init function
- A function implementing the new module state machine, to be called inside the endless loop of the function g3_task_exec

Should the new module request the usage of ST8500 Host Interface commands not to be already managed, the hints provided in Section 7.2 should be applied. Otherwise, the helper functions defined in the hi_msg_impl.c file and the ST8500 Host Interface message parser functions already defined, should be amended to support the new module.

### 7.2 Managing ST8500 G3-PLC Host Interface messages

The addition of new features and new tasks could require that the user implements additional helper functions to ease the preparation of commands to be sent to, and to decode the messages coming from, the ST8500 G3-PLC Host Interface. To do that, the user should modify hi_msgs_impl.c with the required functions.

As an example of how the user should proceed, the implementation of the command G3ICMP_ECHO.Request is described. This command executes an ICMP echo request to a specific node. The format of the Host Interface command implementing this feature (as described in AN5603 - ST G3-PLC Hybrid solution) is shown in Table 2.

<table>
<thead>
<tr>
<th>#Bytes</th>
<th>Label</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>DESTADDR</td>
<td>-</td>
<td>The 16 bytes IPv6 remote destination address</td>
</tr>
<tr>
<td>1</td>
<td>HANDLE</td>
<td>0x0-0xFF</td>
<td>Handle returned in the G3ICMP-ECHO.Confirm</td>
</tr>
<tr>
<td>2</td>
<td>DATALEN</td>
<td>0-1500</td>
<td>The ECHO request payload length</td>
</tr>
<tr>
<td>0-1500</td>
<td>DATA</td>
<td>-</td>
<td>The ECHO request payload</td>
</tr>
</tbody>
</table>

The helper function which compiles the command is the hi_ipv6_echo_req_set, which is reported here below. This function accepts as parameters:

- `msg_`, the pointer to a buffer which contains the command sequence to be sent to the ST8500 Host Interface
- `dst_addr`, the IPv6 address of the destination node
- `handle`, a user-defined ID which should be used to correlate the request with the answer
- `data_len`, the length of the payload associated with the echo message
• **data**, the buffer containing the payload associated with the echo message

An example of how this helper function should be used is given in the function `g3_ka_panc_ping_device`. The `hi_ipv6_echo_req_set` prepares the message inside the `G3_request_ka.icmp_data_req` structure and returns the length of the prepared message.

Afterwards, the `taskCommPut_user_g3` function is called using the proper command ID macro (`HIF_ICMP_ECHO_REQ`), the pointer to the `G3_request_ka.icmp_data_req` structure and the calculated length as parameters. The message is therefore sent to the ST8500 Host Interface.

To manage incoming messages from the ST8500 Host Interface, such as indications or confirm messages, the user should extend the function `g3_msg_handler`, adding a new handler, or extending one of the existing handlers if the purpose is to extend an existing functionality. Four handlers (one for each implemented module plus the one for the PAN Sorting) are already defined. Each handler is called only if the received message is needed by that functionality (for instance, `g3_app_conf_msg_needed` returns true if `g3_app_conf_msg_handler` needs to process the message in question). Moreover, in case of a confirmation message, the `semConfirmation` semaphore must be released. If the message is to be processed also by the user task, `UserG3_MsgNeeded` must return true for the command ID of the received message. Then, the `taskCommPut_g3_user` function forwards the message to the User task.

### 7.3 Adding a user-defined task

The user can easily add one or more user-defined tasks to this example and the way the G3-PLC task has been implemented should be an example on how the customer should proceed. In general, in case the user-defined task needs to make use of some of the features of the G3-PLC Hybrid PLC & RF protocol, the user should:

• Extend the support of ST8500 G3-PLC Host Interface commands (see Section 7.2).

• In case the communication between the tasks involves the exchange of a large amount of data, implement a memory pool infrastructure (or use the CMSIS_V2 `mem_pool` feature).
8 STM32 SPI Flash programming

To download the RTE and PE images to the ST8500 when it is in the BOOT FROM UART mode, first, it is necessary to store such images on the STM32 SPI Flash. The images can be acquired by transferring them with the UDP Image Transfer (see Section 5.6.3) or by using a specific External Loader inside STM32Cube Programmer named M25P16_EVLKST8500GHxxx. This External loader sees the STM32 SPI Flash memory space mapped from the virtual address 0x90000000 to virtual address 0x901FFFFF.

Figure 9. STM32Cube Programmer External loaders menu

To add the M25P16_EVLKST8500GHxxx external loader to the list inside STM32CubeIDE, copy and paste the M25P16_EVLKST8500GHxxx.stldr file inside the \STMicroelectronics\STM32Cube\STM32CubeProgrammer\bin\ExternalLoader folder. Once the file has been copied, follow these steps to program the STM32 SPI Flash memory:

1. Open STM32CubeProgrammer.
2. In the External loaders menu, select the M25P16_EVLKST8500GHxxx external loader.
3. Connect to the STM32 via ST-LINK.
4. In the Memory & File edition menu, open the RTE image file `<filename>.img.bin`, change its address to 0x90000000, and then click "Download".

Figure 10. STM32Cube Programmer Memory & File edition menu

5. In the Memory & File edition menu, open the PE image file `<filename>.img.bin`, change its address to 0x90040000, and then click "Download".

6. Disconnect from the STM32.

Once this procedure has been completed, the STM32 SPI Flash memory shall contain the RTE image at address 0x0 and the PE image at address 0x40000.
Revision history

Table 3. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>19-Oct-2021</td>
<td>1</td>
<td>Initial release.</td>
</tr>
<tr>
<td>15-Mar-2022</td>
<td>2</td>
<td>Revisited document for the V1.5.0 software package.</td>
</tr>
</tbody>
</table>
# Contents

1 G3-PLC Hybrid PLC & RF software solution overview .............................................. 2
   1.1 What is the G3-PLC Hybrid PLC & RF? ............................................ 2
   1.2 Protocol basics ................................................................ 2
   1.3 Supported Hardware and evaluation boards ........................................ 3

2 The G3-PLC Hybrid PLC & RF software package .............................................. 5

3 STM32CubeMX project .................................................................................... 6
   3.1 Project description ....................................................................... 6
   3.2 FreeRTOS subsystem ................................................................ 7
   3.3 UART interface modules ................................................................ 9
       3.3.1 ST8500 Host UART Interface .............................................. 10
       3.3.2 User UART Interface ........................................................... 10
       3.3.3 ST8500 Host UART Pass-through .......................................... 10
   3.4 Task communication mechanism ...................................................... 11
   3.5 ST8500 image download ............................................................... 12

4 G3-PLC applications ................................................................................... 13
   4.1 Introduction ............................................................................... 13
   4.2 The G3-PLC Config module .......................................................... 13
       4.2.1 Attributes table initialization ................................................ 13
       4.2.2 G3-PLC platform configuration ............................................. 13
   4.3 G3-PLC attributes configuration ...................................................... 14
   4.4 The G3-PLC Boot module .............................................................. 14
       4.4.1 PAN device implementation .................................................. 14
       4.4.2 PAN coordinator implementation .......................................... 14
   4.5 The G3-PLC Keep-alive module ...................................................... 15
       4.5.1 PAN device implementation .................................................. 15
       4.5.2 PAN coordinator implementation .......................................... 15
   4.6 The G3-PLC task .......................................................................... 16

5 User applications ........................................................................................... 17
   5.1 Introduction ............................................................................... 17
   5.2 User task interface with the G3-PLC task .......................................... 17
### 5.3 User task interface with the serial terminal
- **5.3.1** Reception (from terminal to User task) ........................................ 17
- **5.3.2** Transmission (from User task to terminal) ..................................... 17

### 5.4 User task internals
- **5.4.1** G3-PLC task communication management .................................... 17
- **5.4.2** User Interface FSM ..................................................................... 18

### 5.5 Use of serial terminal ....................................................................... 18

### 5.6 UDP application example
- **5.6.1** UDP tests .................................................................................. 19
- **5.6.2** UDP connection setup .............................................................. 20
- **5.6.3** UDP image transfer ................................................................. 20
- **5.6.4** SPI Flash management ............................................................. 21
- **5.6.5** Custom commands ................................................................. 21

### 6 Print applications ............................................................................. 22
- **6.1** Introduction .................................................................................. 22
- **6.2** Print task implementation .............................................................. 22

### 7 Extending the G3-PLC Hybrid PLC & RF software example .......... 23
- **7.1** Extending G3-PLC task with additional features ......................... 23
- **7.2** Managing ST8500 G3-PLC Host Interface messages ..................... 23
- **7.3** Adding a user-defined task ............................................................ 24

### 8 STM32 SPI Flash programming .......................................................... 25

### Revision history ................................................................................. 27

### Contents ............................................................................................ 28

### List of tables ....................................................................................... 30

### List of figures ..................................................................................... 31
List of tables

Table 1. ST8500 G3-PLC Hybrid PLC & RF supported boards ........................................... 4
Table 2. G3ICMP_ECHO.Request command format ......................................................... 23
Table 3. Document revision history ........................................................................... 27
List of figures

Figure 1. G3 Hybrid PLC & RF protocol stack ...................................................... 2
Figure 2. Hybrid network example ............................................................. 3
Figure 3. ST chipset implementing the G3-PLC Hybrid PLC & RF protocol stack ........................................................................ 3
Figure 4. STM32CubeMX project initial view ...................................................... 6
Figure 5. STM32CubeMX project peripheral configuration view ........................................ 7
Figure 6. G3-PLC Hybrid PLC & RF SW package architecture .......................................... 8
Figure 7. STM32CubeMX FreeRTOS configuration panel ............................................. 9
Figure 8. Typical test setup using the PC serial terminal ............................................. 19
Figure 9. STM32Cube Programmer External loaders menu ........................................... 25
Figure 10. STM32Cube Programmer Memory & File edition menu ............................ 26
IMPORTANT NOTICE – READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgment.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2022 STMicroelectronics – All rights reserved