IT700/IT900 Host Interface SW Example Application Note

Table of Contents

1 References

No.	Serial	Document Name
1.	*IT700-UM-001-R2.0*	*IT700 Host Interface Command Set User Guide*
2.	*IT900-UM-001-R1.2*	*IT900 Host Interface Command Set User Guide*

2 Scope

The host application example software is a reference code for development of application running on top of the IT700/900 PLC modems. This document describes the PLC modem host application example software code, its structure and functions. The purpose of this application example is to serve as a reference for the development of various PLC applications.
This document assumes knowledge of the IT700/900 Host Interface Commands Set User Manual document.

3 General Description

3.1 Application General Description

The application implements the example of an Automatic Meter Reading system (AMR). The AMR system contains two types of stations:

- Remote Meter Station (RS) – emulates the real Meter by running the increasing counter.
- Network Concentrator Station (NC) – responsible of reading all meters' values in the network.

Both applications use Network Layer management of IT700/900 to create a tree type network topology.
The NC application implements the following functions:

- Initialization of the PLC modem.
- Handling NL (Network Layer) Management indications regarding network status (Network ID assigned) and new station admission (Admission Request).
- Polling Meter values according to the current topology.

The RS application implements the following functions:

- Initialization process of the PLC modem.

- Handling NL Management indications regarding node status (Connected/ Disconnected from Concentrator).
- Sending values from the meter to the concentrator upon request via power line.

3.2 Software Environment Description

The software was burned onto an ATMEGA64 µC from ATMEL. The software is a single thread written in ANSI C and requires no RTOS resources or services.

3.3 Software Block Diagram

Figure 1 shows the block diagram of the applications:

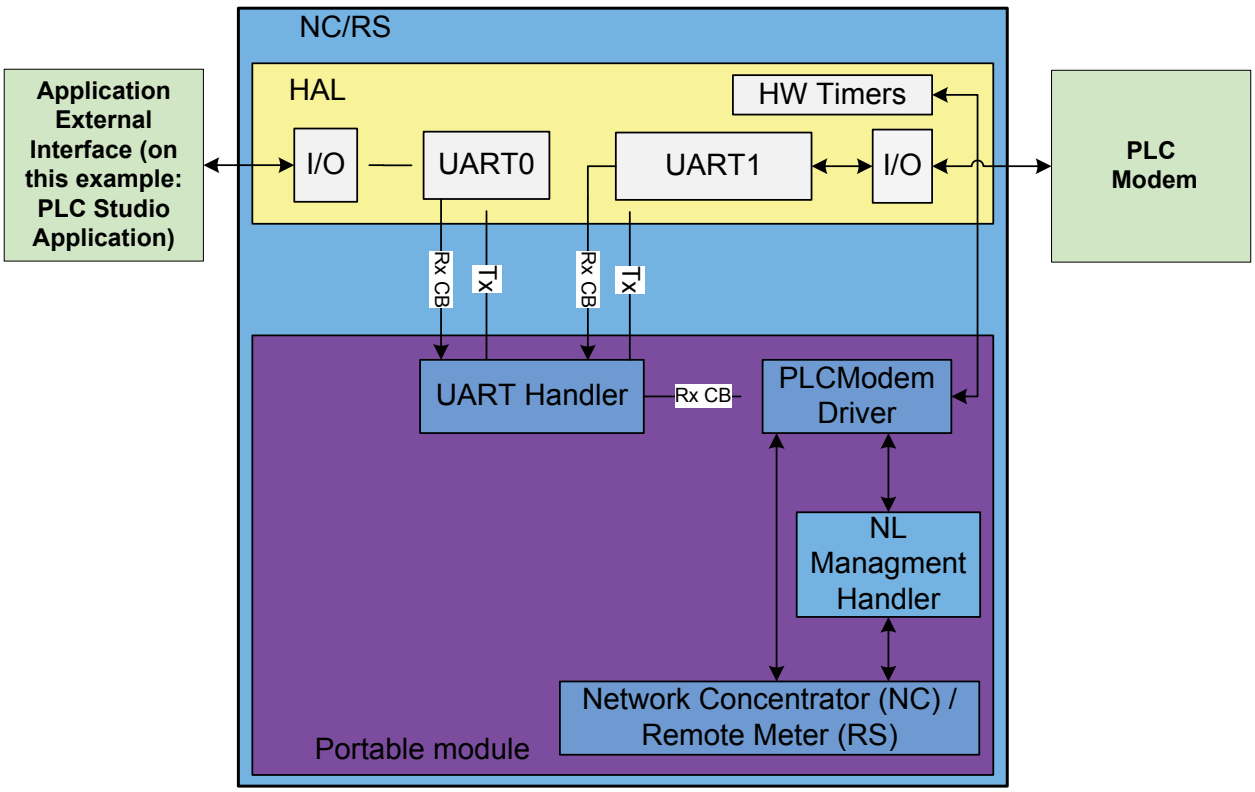


Figure 1: Application Block Diagram

The application is divided into two parts:

- HAL (Hardware Abstraction Layer) – this part of the code is microprocessor dependant and should be implemented for each microprocessor, separately. The HAL module contains:
  - Timer module – implements timer functions.
  - UART module – implements the UART interface to the IT700/900 modem.

  - I/O module – implements the input / output functions to the microprocessor pins.
- Portable module – this part of the code does not depend on the microprocessor. The portable module contains:
  - PLC Modem Driver – implements an interface to IT700/900 Configuration commands, Stack commands and Data commands and provides functions for modem Reset, Packet Reception, Packet Transmission, Parameters Configuration (including Remote Configuration), Parameters Monitoring, Go On Line and more.
  - UART Handler – implements transparent tunnel which is mainly used for running standard IT700/IT900 debug tools (as PLC Studio) through application controller external interface. See more details on section 5.2.4.
  - NLMng Handler – implements an interface to the PLC modem NL Management commands and provides functions for setting Operation Mode, Admission Mode and Network Size, handling Network Status Indications and more.

3.4 Package Structure

The software example contains the following folders:

- Docs – the folder with all package documentation:
  - *Host Interface SW Example Application Note* (current doc)
- Code – the folder contains all application example codes:
  - HAL: the folder contains all HAL related files:
    - HAL_Timer.c, HAL_Timer.h – Timer module implementation.
    - HAL_UART.c, HAL_UART.h – UART module implementation.
    - HAL_Portability.h – data types definitions.
    - HAL_IO.c, HAL_IO.h – pin out definitions and functions.
    - HAL_Init.c, HAL_Init.h – HAL initialization module.
  - PI:
    - NLMngHandler.c, NLMngHandler.h – handles the interface to the NL Management (described in reference 1)
    - PLCModemDriver.c, PLCModemDriver.h – handles the interface to PLC modem Configuration commands, Stack commands and Data commands (described in reference 1)
    - UARTHandler.c, UARTHandler.h – handles the interface to external applications as the PLC Studio Application (as described in section 5.2.4.

- Main.c – implements both the Network Concentrator and Remote Meter function (depending on compilation definitions).

4 Processes Description

4.1 Initialization Process

The initialization process consists of two phases:

- Application Modules initialization.
- PLC Modem initialization.

After the initialization steps are completed, the application enters the main loop.

4.1.1 Modules Initialization

- Init. HAL module
- Init. PLC Driver module
- Init. NLMng Handler module
- Init. UART Handler module
- Set PLC Modem baud rate – this step is required when using IT900 only:
  When using the IT900 modem and the ATMEGA64 µC, the ATMEGA64 baud rate (38400bps) is slower than the IT900 baud rate (921600bps). Therefore, their baud rates must be coordinated. On this example, the application sets the IT900 to low baud rate mode (38400bps) by clearing the I/O PD1 (of the ATMEGA64) which is connected to the IT900 UBRATE I/O. If you are using different µC which does support IT900 fast baud rate (921600bps), there is no need to set the IT900 to low baud rate mode. Another option is to set the desired default baud rate in factory defaults on manufacture stage.
- Set the callback function for the PLC Modem Received Packets

4.1.2 Modem Initialization

- Hard reset the PLC modem.
  Note: In case the µC (ATMEGA64 in this example) power-up wakeup time is shorter than the PLC Modem power-up wakeup time, then before sending the first command to the PLC Modem, a short delay must take place (as can be seen at the beginning of the main function). This will assure the application uC will get the first reset response from the PLC Modem after the hardware reset.
  Notice that the delay is variable, and it depends on which micro-controller and power supply is being used.
- Wait for the Reset response from the modem (Notice that resetting the modem is necessary to start working. The application won't continue till it gets a success reset response).
- Configure relevant parameters:

- NC:
  - Set Operation mode to “Base”
  - Optional (shown in comments): Set admission mode to “Application approval”
- RS:
  - Set Operation mode to “Adaptive”

- Send Go Online command and wait for response from the modem.

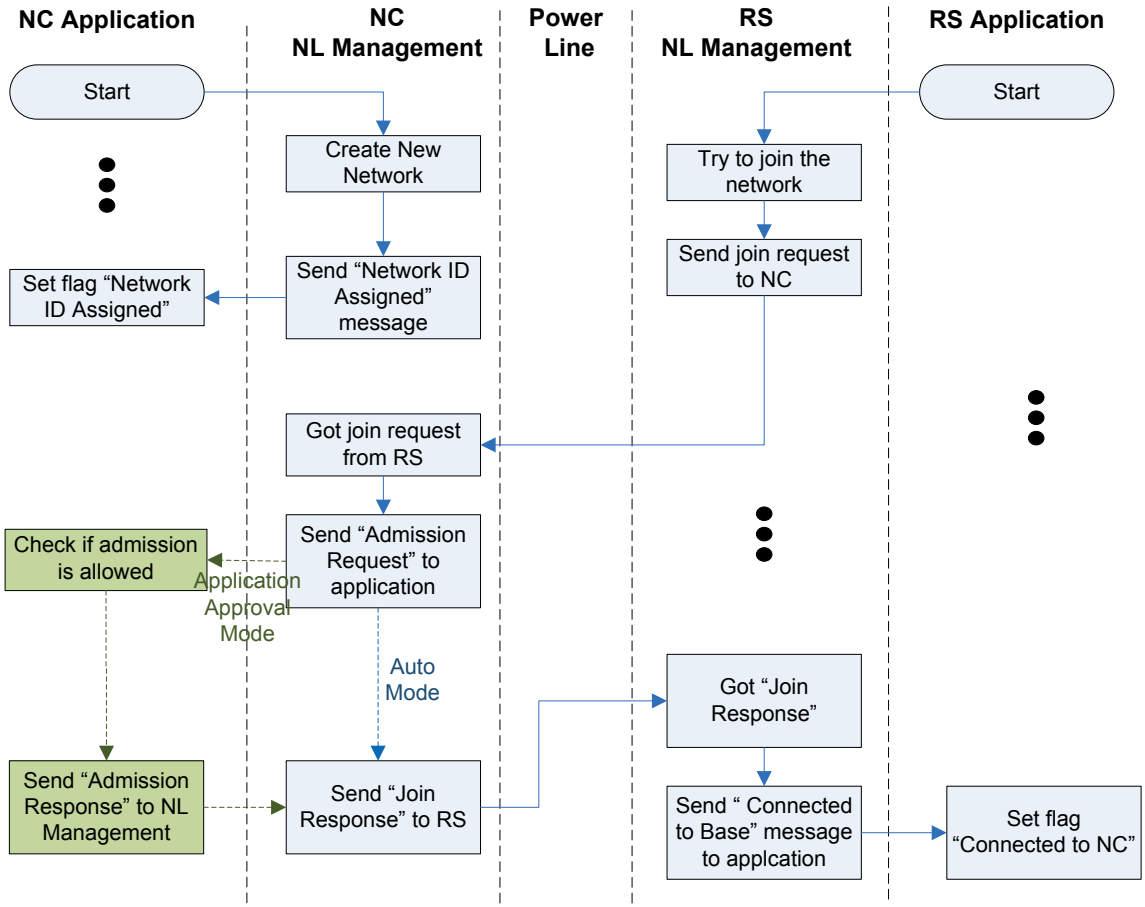
4.1.3 Network Creation

Once the NL operation mode is configured as “Base” in the NC application (see Section 4.1.1), the NL Management automatically creates a new network. After the Network ID is selected, the NL Management module will send a “Network ID Assigned” message to the application. The NC application may poll the stations in the network only after the Network ID is assigned.

Once the NL operation mode is configured as “Adaptive” in the RS application (see Section 4.1.1), the NL Management will automatically try to join a network. When the admission mode is set to “Application approval” (this is optional, and it’s shown in comments within the code), the NC application will receive the “Admission request” to approve the joining of the remote station to the network during the process. When the join process is completed successfully, the RS application will receive a “Connected to NC” message from its NL Management module. The RS application may reply to meter request messages from the NC only after the station is connected to NC.

The network creation process is shown in Figure 2. When the admission mode is set to "Auto" (default), the green blocks and the dashed green lines are not part of the admission process. When the admission mode is set to "Application approval" (this is an optional code section marked as comment), the green blocks and the dashed green lines are taking place while the dashed blue line is not part of the admission process.

Figure 2: Network creation process



4.2 PLC Modem Interface

The PLC Modem driver module implements an interface to the IT700/900 (described in Reference 1 and 2 respectively).

The PLC Modem driver uses UART HAL functions to communicate with PLC Modem and implements the following processes:

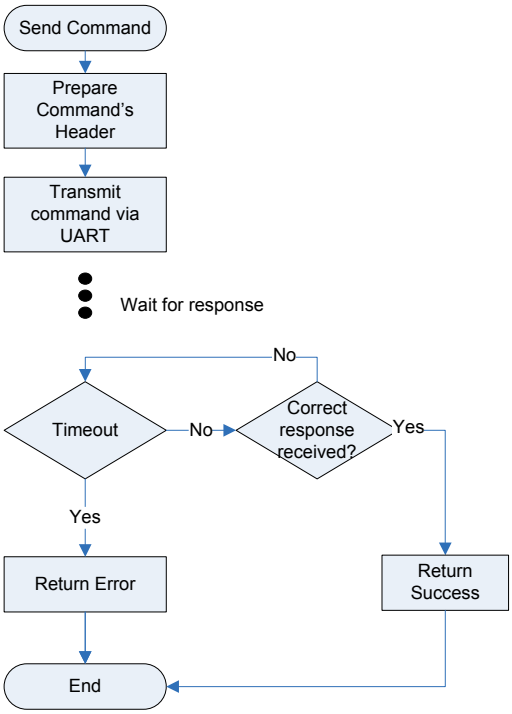
- Send data to PLC Modem
- Receive data from PLC Modem

4.2.1 Send Data to the PLC Modem

4.2.1.1 Send command to PLC Modem

The process of sending a command to the PLC Modem is shown in Figure 3:

Figure 3: Send command to PLC Modem process

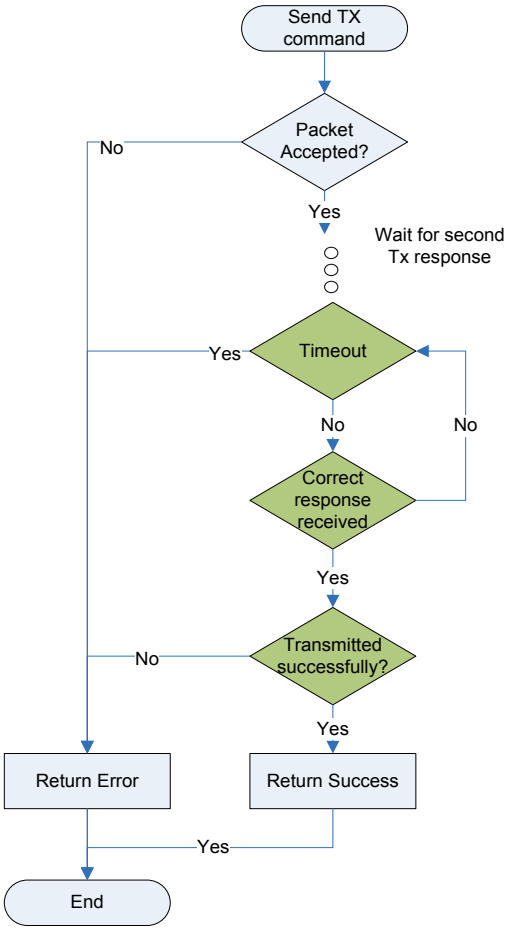


It is forbidden to send another command before the application receives a response from the PLC Modem or before the timeout passed.

4.2.1.2 Transmit Data Packet to the Powerline

After sending a command to the PLC Modem, the first response must be received (see Section 4.2.1.1). When the command is accepted for transmission, the application can wait for a second response (this is optional, and waiting for second Tx response is marked in a comment in the code) from the PLC Modem regarding transmission status (the packet successfully transmitted, no acknowledgement was received, etc.). The second part of the transmission command is shown in Figure 4 (the optional part is marked in green):

Figure 4: Transmit data packet to PLC



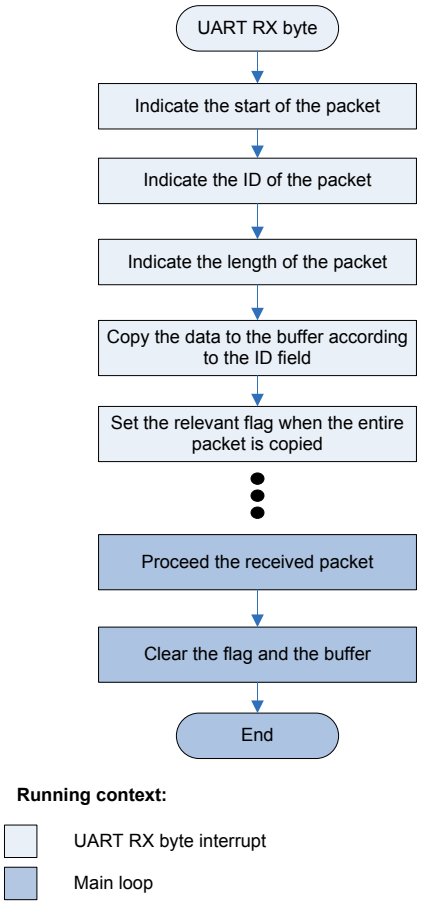
4.2.2 Receive Data from the PLC Modem

The receive data process for the PLC Modem driver module uses the UART interrupt callback function when a new byte is received. Three data types may be received from the PLC Modem:

- Response to the command (see Section 4.2.1.1).
- Data packets received from the power line.
- Indication from the NL Management.

Since all three data types can be received by applications one after another without any delay, the PLC Modem driver module contains three different buffers for each type of data from the PLC Modem. Incoming data is copied to the appropriate buffer according to the Type and Opcode fields. The received data then proceeds in the main loop. The flow of this process is shown in Figure 5:

Figure 5: Reception process from PLC Modem



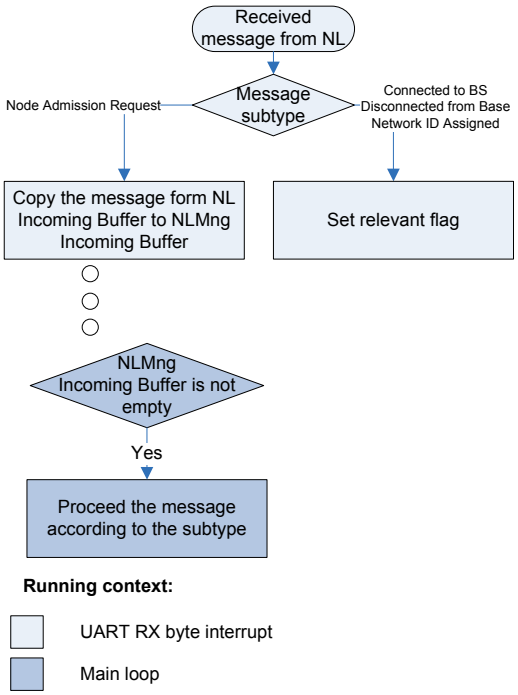
4.3 PLC Modem NL Management Interface

The NLMngHandler module implements the interface to the NL Management functions. The NLMngHandler module uses an interface to the PLC Modem driver module of the application for communication with the modem to receive indications from the NL Management.

4.3.1 Receive Indication from the NL Management

Any message from the NL Management is copied by the PLCModemDriver module to the NL Incoming Buffer (see Section 4.2.2). After the message is received, the *NLMngHandler_MessageHandler* function is called. If the message processing is simple (set flag, etc.) the process is immediate, otherwise the message is copied to the NLMng Incoming Buffer for further processing in the main loop. This mechanism allows receiving two NL Management indications with no delay between them.
The process is shown in Figure 6:

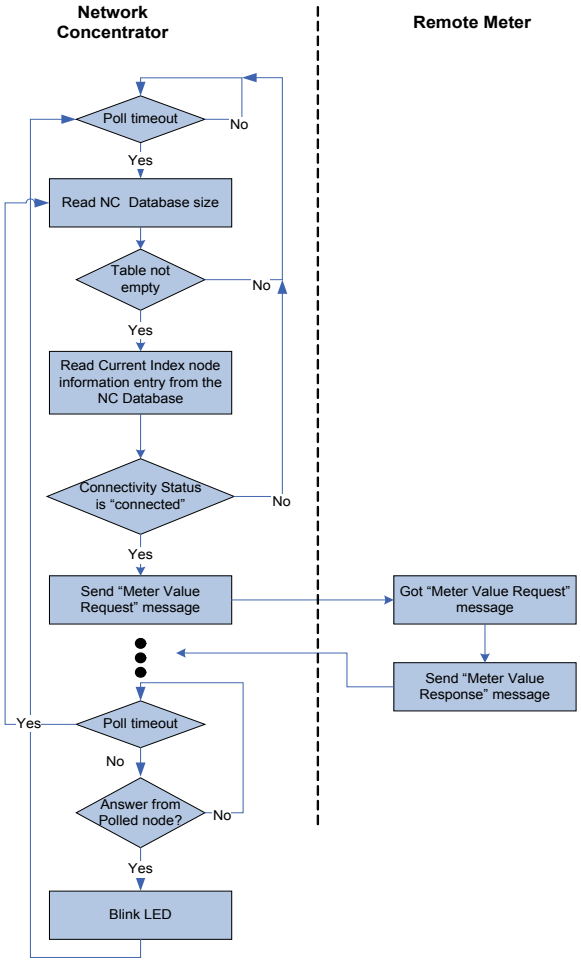
Figure 6: Receive message from NL Management process



4.4 Remote Meter Values Polling Process

The NC polls meter values from all Remote Meter Stations in the network. The NC reads the NC Database to get the next Meter Node ID; sends a request to the meter and waits for a response. When the Remote Meter station receives a request from the NC, it sends a response with the current meter value (counter value). If the response is received before the next poll timeout, the NC debug LED will blink.
This process is shown in Figure 7:

Figure 7: Meter value poll process



5 Software Detailed Description

5.1 Hardware Abstraction Layer (HAL)

The HAL module implements the Timer, UART and I/O functions for the AMR microprocessor. This document does not detail HAL implementation. However, all HAL files are provided within the software example. The HAL interface functions that must be implemented in order to use the portable code with no changes are described in section 6. For further information, contact Yitran Customer Support.

5.2 Portable Module

5.2.1 Main Function

The main function implements the following:

- Initialization process (see Section 4.1) – calls the initialization functions of the HAL, PLCModemDriver, NLMngHandler and UARTHandler modules and implements the *InitModem* function, which is slightly different for the NC and the RS (different configuration parameters).
- Main loop that is entered when the POLL_TICKS timer ticks (compilation defined). If a timeout occurs, the main function performs the following tasks:
    - Calls the *NLMngHandler_ManageNLMng* function to handle received events from the NL Management.
    - Calls the *PLCModemDriver_ManageModem* function to handle received events from the PLCModemDriver module.
    - Implements application dependant function?. In the current example, only the NC application uses this to implement the remote meter poll mechanism (see Section 5.2.1.1)
    - Starts the timer for the next poll timeout.
    - Interface Activity monitoring – two options are provided as reference for implementation of interface activity monitoring:
        1. NOP command is sent to the modem.
        2. Validation of normal behavior of PLC Modem activity pin I/O.

      The Interface Activity monitoring operation does not need to be activated when using IT700/900 PIMs manufactured by Yitran and is disabled on default in the example code.

5.2.1.1 NC (Network Concentrator)

The network concentrator contains the following functions:

- *NC_SendNodeRequest* - the function sends the request to remotely? read meter values:
    - Input: the Node ID of the meter to send the request to.
    - Output: void.
- *NC_PollNextMeter* – the function starts the poll meter values process (see Section 4.4) and is called from the main loop.
    - Input: void.
    - Output: void.
- *NC_CB_OnIncomingDataPkt* – the function is the callback function when the data packet is received by the PLCModemDriver module. In NC the data packet can only be the remote meter's response to the NC poll request. If the Node ID of the original source of the response is equal to the Node ID of the last request sent, the NC blinks the debug LED.

- Inputs:
  - Pointer to the received buffer.
  - Buffer length.
  - The Node ID of the original sender (RS).
- Output: void.

5.2.1.2 RS

The Remote Meter Station implementation contains the following functions:

- *RS_SendValue* – the function sends the current meter value to the NC:
  - Inputs:
    - The Node ID of the NC to send the value to.
    - The current meter value.
  - Output: void.
- *RS_CB_OnIncomingDataPkt* – the callback functions when a data packet is received by the PLCModemDriver module. In the RS it may be only a request from the NC. The function sends the current meter value to the NC if the following conditions are true (1) the received message is the valid request from the NC and (2) the remote station has the valid connection to the NC:
  - Inputs:
    - Pointer to the received buffer.
    - Buffer length.
    - The Node ID of the original sender (NC).
  - Output: void.

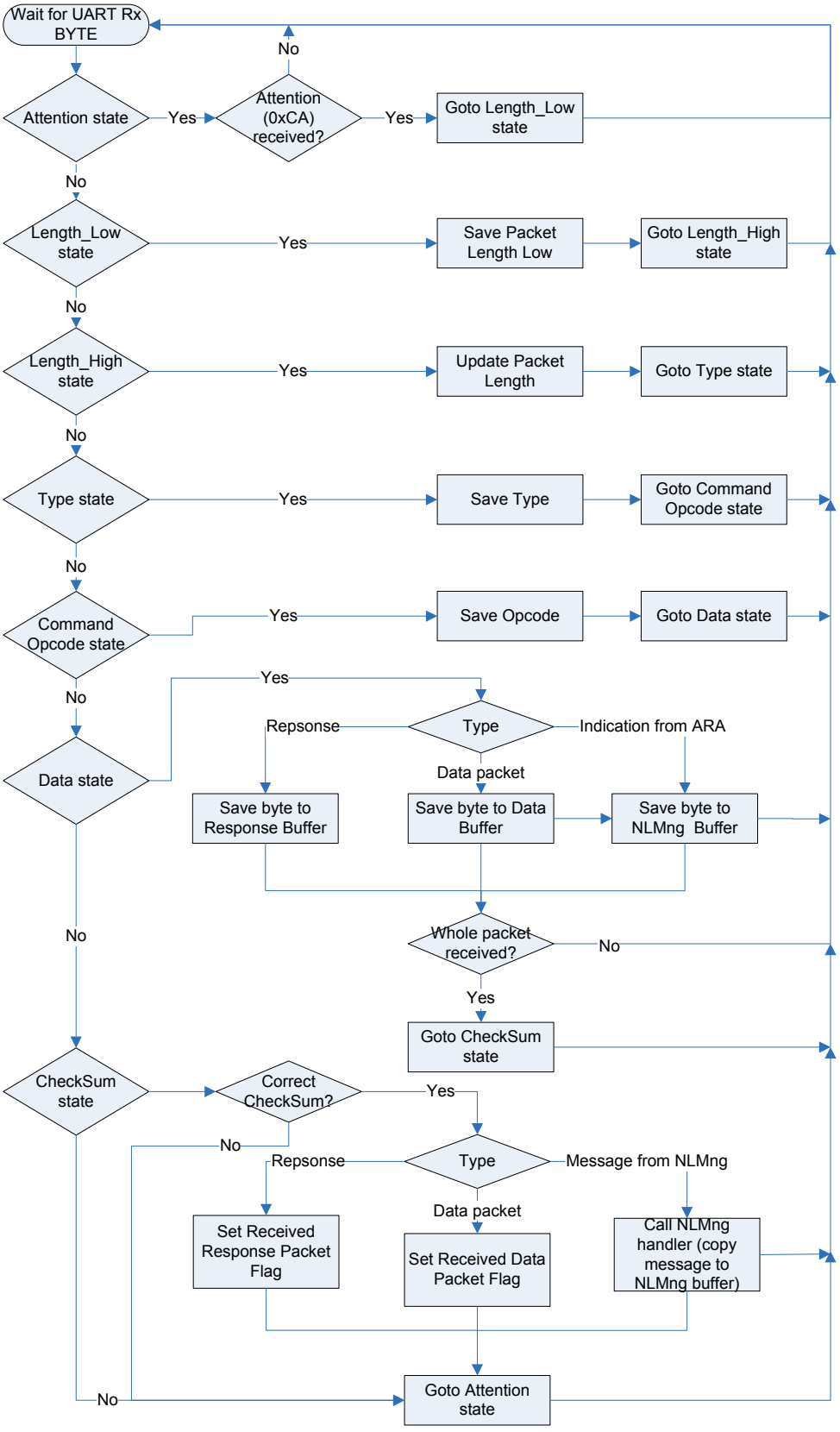
5.2.2 PLC Modem Driver

The PLC Modem driver module implements the interface to the PLC modem. The main functions of the module include:

- Timeout functionality – There are several modules in the example that require the timeout function. The example provided uses the single hardware timer for this purpose using the following functions:
  - *PLCModemDriver_wCalculateTimeoutValue* – receives the required number of timer ticks and calculates the value that the hardware timer should reach for the timeout (current timer value plus the required number of ticks):
    - Input: Required number of ticks until next timeout.
    - Output: The value of hardware timer for the timeout to occur.
  - *PLCModemDriver_bIsTimeoutOccured* – returns? TRUE when the timeout occurs:
    - Input: The value of hardware timer for the timeout to occur (an output of the *PLCModemDriver_wCalculateTimeoutValue* function).

    - Output: TRUE or FALSE.
- *PLCModemDriver_CB_OnRxByte* – the callback function from the UARTHandler module when a new byte is received (see Section 4.2.2) as shown in Figure 8.
- There are three different buffers for the received data, and *PLCModemDriver_CB_OnRxByte* function calls different functions to fill different buffers:
    - *PLCModemDriver_FillRspBuffer* function when the response from the PLC Modem is received.
    - *PLCModemDriver_FillRxDataBuffer* function when the data packet is received from power line.
    - *PLCModemDriver_FillNLBuffer* function when the data from the NL Management is received.

Figure 8: On UART RX byte callback function



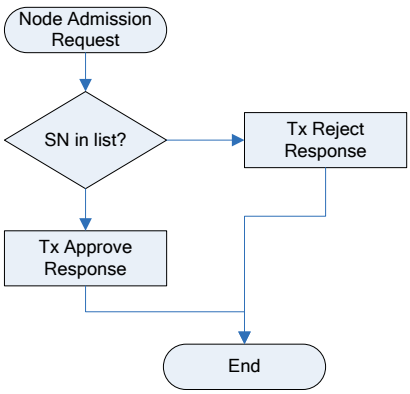
- *PLCModemDriver_ManageModem* – handles the different messages from the PLC Modem. If the reset response is received, the *PLCModemDriver_bGoOnline* function is called to start the modem. If a data packet is received, the application callback function (different for NC and RS) is called. It also deals with remote configuration change indication.
- *s_bSendWaitForResponse* – sends a command to the PLC Modem using the HAL_UART module and waits for an immediate response (see Section 4.2.1.1). If the timeout occurred, the function returns FALSE, otherwise it returns TRUE.
    - Input: pointer to the buffer with the command to send and its length.
    - Output: TRUE or FALSE.
- The PLC Modem driver module implements various commands using the *s_bSendWaitForResponse* function. All commands prepare the buffer with the command information, as described in the interface documents, and waits for the immediate response from the modem. This process ensures no additional command will be send to the modem before an immediate response is received. The module implements the following functions:
    - *PLCModemDriver_TransmitPacket* – send command to transmit the data packet to the power line.
    - *PLCModemDriver_bGoOnline* – send Go Online command.
    - *PLCModemDriver_bGetParam* – read parameter (configurable parameter, factory default, factory default validity, debug counters).
    - *PLCModemDriver_bSetParam* – set configurable parameter / set factory default.
    - *PLCModemDriver_bGetSerialNum* – get station's serial number.
    - *PLCModemDriver_bSetSerialNum* – set station's serial number..*?*
    - *PLCModemDriver_bSaveParameters* – save configurable parameters in the non volatile memory.
    - *PLCModemDriver_bGetNcDatabaseSize*– read the size of NC Database (nodes information).
    - *PLCModemDriver_bGetNodeInfo* – read the entry from NC Database according to the given index.
    - *PLCModemDriver_bRemoteSetParam* – implements the Remote Configuration function. The function is not used in the code and provided as an example only.

5.2.3 NLMng Handler

The NLMng handler module implements the interface to the NL Management commands (see Section 4.3) and the handler of the received indication messages from the NL Management. The module contains the following functions:

- *NLMngHandler_MessageHandler* – the function is called from the PLC Modem driver module when the message from NL Management is received. The function sets the relevant flag (like when Connected To Base message is received in the RS) or copies the buffer to the sbyNLMngIncomingBuffer for further processing in the main loop.
- *NLMngHandler_ManageNLMng* – the function is called from the main loop and handles the NL Management messages (like admission request in the NC application).
- *CheckAdmission* – the next process is irrelevant when the "Set Admission Mode" part is marked as a comment within the code (in *InitModem* function). This function is relevant for the NC application and is called when the NL Management gets the request from the remote meter station to join the network of the specific NC. The process of the admission approval is shown in Figure 9. For the successful approval the Serial Number of the remote station must exist in the predefined sbyApprovedSNs array.

Figure 9: Admission approval in NC.



5.2.4 UART Handler

The UART_HANDLER module implements transparent tunnel mainly used for running standard IT700/IT900 debug tools (such as PLC Studio) through application controller external interface. The motivation of using the UART Handler is to enable the access to IT700/IT900 host interface through the application controller external interfaces.

Every Rx interrupt from one of the UARTs, calls the *UART_Handler_CB_OnRxByte* function. If the current state is Application mode (see description of the state machine on Figure 10), then the handling goes to *PLCModemDriver_CB_onRxByte*. If the current state is PLCS Application mode (Studio mode), the *UART_Handler_CB_OnRxByte* will continue handling this state.

The main functions of this module include (UART_Handler.*c*):

- *UART_Handler_CB_OnRxByte (BYTE byData, BYTE byEvent)* – the callback function from the UART0/UART1 modules when a new Rx interrupt is occurred (from both UARTs). This function handles the state machine (see Figure 10).
  The function inputs are *byData* – which is the received byte from one of the UARTs, and *byEvent* - which indicates which one of the UARTs call the function. Accordance with the appropriate event, the function transmits the received byte using the HAL_UART_UART0_Transmit_Byte(byData)/ HAL_UART_UART1_Transmit _Byte(byData) function to UART0/UART1 respectively.
- *Init_UARTHandler()* – the function initialize the UARTHandler module.

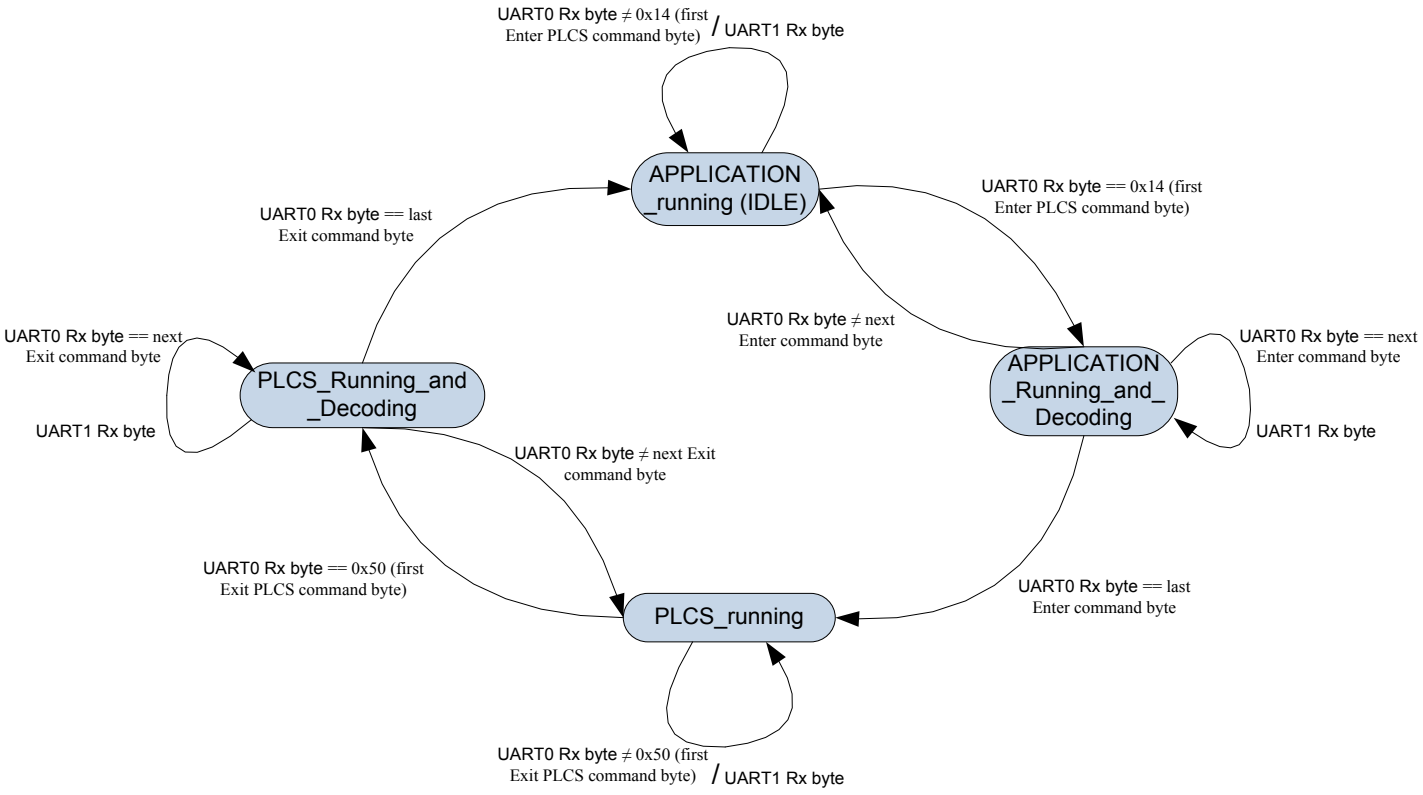


Figure 10 - UART_Handler State Machine

For running PLC Studio through Application controller external interface, the following steps should be applied:

- Open PLC Studio
- In the 'Connect' screen set the 'Enter Studio Mode' checkbox.
- Press the 'Apply' button.

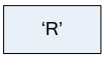
Following this process the Application State Machine will be set to "PLCS_running" state – see Figure 10.
When done using the PLC Studio, the user should return the Application State Machine to " APPLICATION_running " state (see Figure 10). This can be done in two different ways:

1. Reset the application.
2. Send 'Exit' command from the external application: {0x50, 0x28, 0x88, 0x41, 0x97, 0x16, 0x93, 0x99, 0x37,0x51, 0x5, 0x82, 0x20, 0x97,0x49, 0x44}.

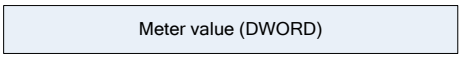
5.3 Packet Formats

The application uses two packet types:

- Meter Value Request – the packet is sent from the NC to the remote meter to get the current meter value. The packet has the following payload:



- Meter Value Response – the packet is send from the RS to the NC when the Meter Value Request packet is received. The packet has the following payload:



5.4 Data Structures

5.4.1 PLC Modem Driver Data Structure

The PLC Modem Driver module contains the following global data structure (called g_PLCModemDriver_struct):

- Pointer to buffer for the response from the PLC Modem.
- Pointer to buffer for the data packets received from power line.
- Pointer to buffer for the messages/responses from the NL Management.
- Pointer to the callback function for incoming data packets.
- RX state – current state of the *PLCModemDriver_CB_OnRxByte* function.
- bResetReceived – the flag that the reset indication was received from the modem.

- byIncomingPacketType - type of the packet being received.
- byIncomingPacketOpcode – opcode of the packet being received.
- byChecksum – checksum of the packet being received.
- wInPLIdx – index of the incoming byte.
- wInLength – expected incoming packet length.

5.4.2 NLMng Handler Data Structure

The NLMng handler contains the following global data structure:

- The buffer for the message received from the NL Management.
- Boolean flags:
  - Connected to NC (used in RS application)
  - Network Address assigned (used in NC application)
  - Admission request received (used in NC application)
- sbyApprovedSNs – Constant table with Serial Numbers of the Remote Meters.

5.5 Compilation Definitions and Constants

Example Code for both NC and RS is implemented on a single file – main.c. To compile RS or NC application, the user needs to define the STATION_TYPE as REMOTE_METER or NETWORK_CONCENTRATOR (in the main.c file) respectively.

In order to create the network, the serial numbers of all remote stations should be defined in the sbyApprovedSNs array? (in the NLMngHandler.c file).

6 Porting to Other Platforms

To port the example to other platforms, create a new hardware abstraction module for other hardware. This example application was created for single thread environment but can be easily ported to multi-thread by using only one running context for the portable part. The portable part does not require changes.

The interface between the HAL module and the portable part contains the following functions that must be implemented in the HAL module.

- *HAL_Init_ConfigureHAL* – the function should initialize all HAL modules:
  - Input: void.
  - Output: void.
- *HAL_IO_togglePin* – the function provides the requested operation on the IO pin according to the input values:
  - Inputs:
    - Port ID
    - Pin number
    - Pin direction (IN/OUT)
    - Action (set, clear, etc)

- Output: void

- *HAL_IO_TurnOnRed* – the function turns ON the debug red LED:
  - Input: void.
  - Output: void.
- *HAL_IO_TurnOffRed* – the function turns OFF the debug red LED:
  - Input: void.
  - Output: void.
- *HAL_IO_TurnOnGreen* – the function turns ON the debug green LED:
  - Input: void.
  - Output: void.
- *HAL_IO_TurnOffGreen* – the function turns OFF the debug green LED:
  - Input: void.
  - Output: void.
- *HAL_IO_SetModemBaudRate* – the function sets the PLC Modem baud rate:
  - Input: void.
  - Output: void.
- *HAL_UART_UART0_Transmit_Byte* – the function transmit one byte through UART0:
  - Input:
    - Byte to transmit.
  - Output: void.
- *HAL_UART_UART1_Transmit_Byte* – the function transmit one byte through UART1:
  - Input:
    - Byte to transmit.
  - Output: void.
- *HAL_UART_UART0_Transmit* – the function sets the callback function for UART0 RX byte interrupt (the callback function should be called by the HAL module every time the new byte is received on UART interface:
  - Input:
    - Pointer to the callback function *UARTHandler_CB_OnRxByte*.
  - Output: void.
-
- *HAL_UART_UART1_Transmit* – the function transmits the buffer to the UART interface:
  - Inputs:
    - Pointer to data
    - Data length (in bytes)
  - Output: void.

- *HAL_UART_UART0_SetOnRxByteCB* – the function sets the callback function for UART0 RX byte interrupt (the callback function should be called by the HAL module every time the new byte is received on UART0 interface:
  - Input:
    - Pointer to the callback function *UARTHandler_CB_OnRxByte*.
  - Output: void.
- *HAL_UART_UART1_SetOnRxByteCB* – the function sets the callback function for UART1 RX byte interrupt (the callback function should be called by the HAL module every time the new byte is received on UART1 interface:
  - Input:
    - Pointer to the callback function *UARTHandler_CB_OnRxByte*.
  - Output: void.
- *HAL_Timer_StartTimer0* – the function starts the hardware timer (used for all timeouts in the example design):
  - Inputs:
    - Start value
    - Timer prescaling.
  - Output: void.
- *HAL_Timer_wGetTimer0Ticks* – the function reads the current value of the hardware timer:
  - Input: void.
  - Output: WORD value of the timer.

Document Control

Rev.	Description	Date
1.0	Creation	18/01/2012

Important Notice

YITRAN Communications (YITRAN) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

YITRAN warrants performance of its products to the specifications applicable at the time of sale in accordance with YITRAN'S standard warranty. Testing and other quality control techniques are utilized to the extent YITRAN deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). YITRAN'S PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE–SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF YITRAN'S PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, and to minimize inherent or procedural hazards the customer must provide adequate design and operating safeguards.

YITRAN assumes no liability for applications assistance or customer product design. YITRAN does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of YITRAN covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. YITRAN'S publication of information regarding any third party's products or services does not constitute YITRAN'S approval, warranty or endorsement thereof.