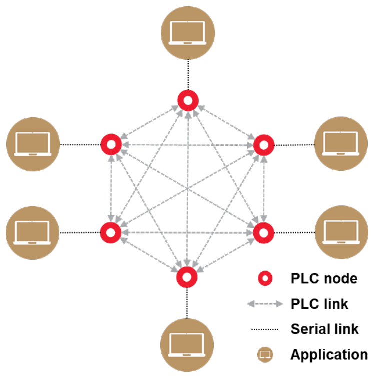
PLC & Go

Introduction

PLC & Go is Microchip's application note to get started quickly with Power Line Communications (PLC). PLC & Go implements a basic application example of low data rate PLC (up to 200 of kbps) using OFDM modulation, that supports point to point and multi-point communications.

PLC & Go runs a chat application between two or more end points using Microchip PLC modem boards. The end points in this example are PCs which are connected via serial interface (USB, UART) to Microchip evaluation boards acting as PLC modems. Any end point can transmit a message, which is received by the other end points in the network. The application example allows end points to configure several PLC parameters such as modulation type, modulation scheme and band of transmission (in case of using a Microchip evaluation board that supports several bands of transmission).



PLC & Go chat application is built on top of the physical layer (PHY) of state-of-the-art standards for PLC, such as PRIME and G3-PLC. As such, PLC & Go is taking advantage of the speed, robustness and frequency band profiles defined by the PHY layers of these standards. For the sake of simplicity, PLC & Go removes all advanced networking features available in the MAC layers of PRIME and G3-PLC. These are not required for the basic point to point / point to multi-point communication scenarios described in this application note. Microchip does provide full PRIME and G3-PLC solutions as well, please check Microchip website for more information. For more information about PRIME and G3-PLC, please check the PRIME Alliance and G3-PLC Alliance websites.

Table 1 below summarizes the Microchip evaluation kits available to run PLC & Go, as well as the communication profiles and band-plans that can be configured.

Table 1. Microchip Evaluation Kits and Supported Frequency Bands

Microchip Eval Kit	Frequency Band	Supported PRIME Profiles	Supported G3 Bandplans
PL360G55CF-EK	[9 … 95 kHz]	PRIME v1.4 channel 1	CEN-A [(1)]
	[148,5 … 500 kHz]	PRIME v1.4 channels 3,4,5,6,7,8	FCC ARIB
PL360G55CB-EK	[95 … 148,5 kHz]	-	CEN-B
ATPL360-EK	[9 … 95 kHz]	PRIME v1.4 channel 1	CEN-A
	[148,5 … 500 kHz]	PRIME v1.4 channels 3,4,5,6,7,8	FCC ARIB
PL485-EK v1	[95 … 148,5 kHz]	-	CEN-B

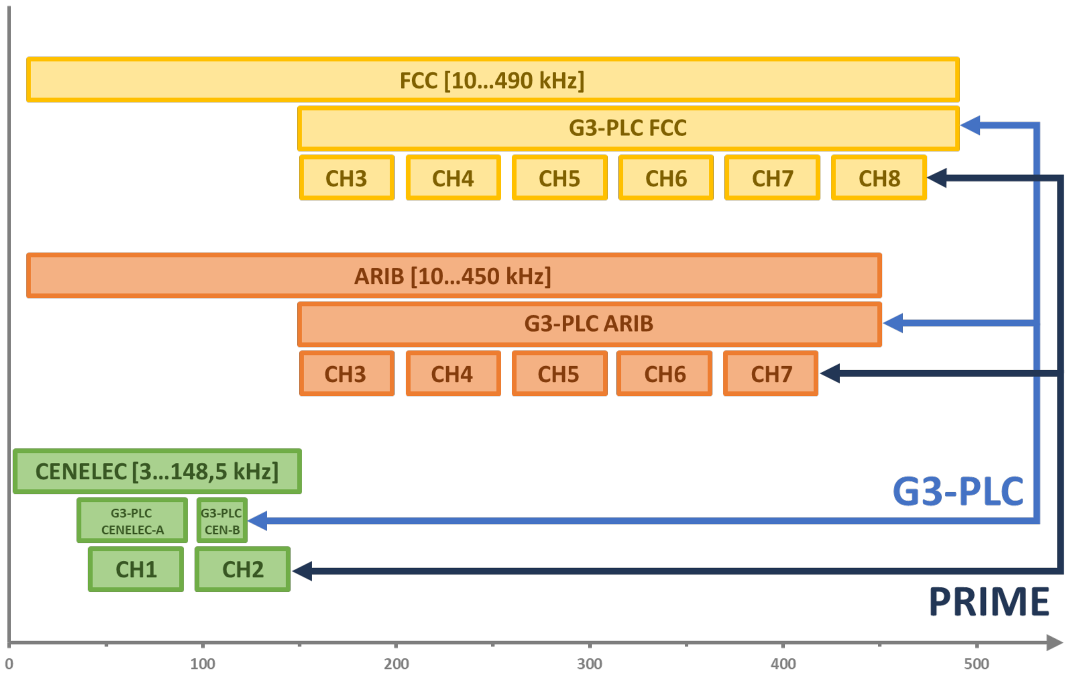
Notes:

1. The hardware of the PL360G55CF-EK board supports several bandplans. The PLC & Go example uses CENELEC-A bandplan by default.

Microchip PLC solution is flexible and provides designs optimized for several frequency sub-bands below 500 kHz. Adopters can decide which band to use based on application's restrictions and /or communications performance, which can be affected by different types of noise sources in the PLC channel.

Figure 1 shows the frequency band profiles that G3-PLC and PRIME provide, following european (CENELEC), north-american (FCC) and japanese (ARIB) regulations.

Figure 1. FCC, ARIB and CENELEC Limits vs PRIME and G3-PLC Bands



Target applications are home and building automation, smart lighting, equipment control over DC lines, solar energy and alarm systems.

Table of Contents

1. Application Example

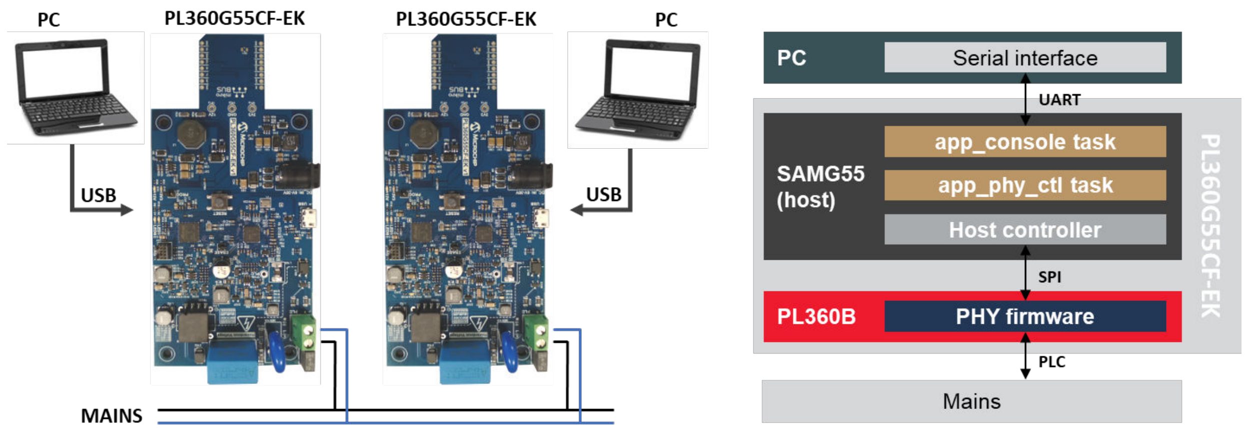
Microchip PLC & Go describes a point to multi-point chat application built on top of the PHY layer of state-of-the-art PLC protocols such as PRIME and G3-PLC.

The application example is part of the Microchip PLC software deliverables available for download from the Microchip website. Note that `phy_plc_and_go` application example can be used with Microchip G3-PLC stack version 1.4.1 or above, and Microchip PRIME v1.4 stack version 14.03.02 or above, in any Microchip platforms implementing PL360 device as PLC modem.

This application example assumes that a host device (a PC, for example) is connected through a serial interface (USB, UART) to a Microchip evaluation kit acting as a PLC modem.

Figure 1-1 below provides an example of the minimum hardware and firmware resources required to run PLC & Go. In this example, two PL360G55CF-EK evaluation kits have been used as PLC modems. Each PL360G55CF-EK board implements a SAMG55 microcontroller, which executes "PLC & Go" application, and a PL360 modem for power line communications, which runs the selected PLC physical (PHY) layer.

Figure 1-1. PLC & Go Block Diagram for PL360G55CF-EK Boards



The PLC & Go application interchanges data with the PC through a serial port using a terminal emulator (see figure Figure 1-2 where terminal emulator TeraTerm is used). The settings of the serial port are: 921600 bps, 8 data bits, no parity, 1 stop bit and flow control.

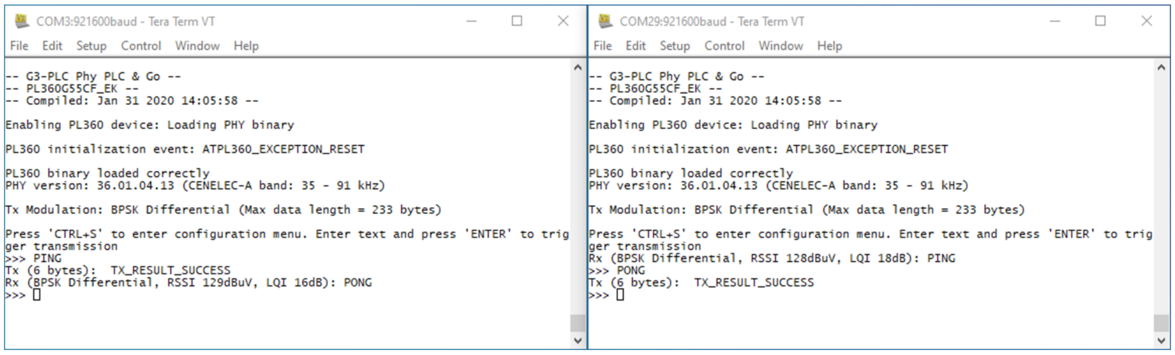
When the serial port is opened, the PLC & Go application shows in the console some information about the hardware platform and the firmware running, and it waits for a message to be sent. The message is sent when a carriage return character is received or the maximum length of the PLC data message is reached. If the user sends the ASCII character 0x13 (‘CTRL+S’), a configuration menu is shown.

The PLC & Go application allows to:

- Configure modulation type and modulation scheme
- Configure the band of transmission (in case of using a Microchip evaluation board that supports several bands of transmission).

In addition, the application provides some information about the transmitted messages (length of the message) and the received messages (Modulation type and scheme, signal quality), which can be displayed by means of a terminal emulator in the PC.

Figure 1-2. PLC & Go Console


COM3:921600baud - Tera Term VT
File Edit Setup Control Window Help

```
-- G3-PLC Phy PLC & Go --
-- PL360G55CF_EK --
-- Compiled: Jan 31 2020 14:05:58 --

Enabling PL360 device: Loading PHY binary

PL360 initialization event: ATPL360_EXCEPTION_RESET

PL360 binary loaded correctly
PHY version: 36.01.04.13 (CENELEC-A band: 35 - 91 kHz)

Tx Modulation: BPSK Differential (Max data length = 233 bytes)

Press 'CTRL+S' to enter configuration menu. Enter text and press 'ENTER' to trig
ger transmission
>>> PING
Tx (6 bytes):  TX_RESULT_SUCCESS
Rx (BPSK Differential, RSSI 129dBuV, LQI 16dB): PONG
>>> 
```

COM29:921600baud - Tera Term VT
File Edit Setup Control Window Help

```
-- G3-PLC Phy PLC & Go --
-- PL360G55CF_EK --
-- Compiled: Jan 31 2020 14:05:58 --

Enabling PL360 device: Loading PHY binary

PL360 initialization event: ATPL360_EXCEPTION_RESET

PL360 binary loaded correctly
PHY version: 36.01.04.13 (CENELEC-A band: 35 - 91 kHz)

Tx Modulation: BPSK Differential (Max data length = 233 bytes)

Press 'CTRL+S' to enter configuration menu. Enter text and press 'ENTER' to trig
ger transmission
Rx (BPSK Differential, RSSI 128dBuV, LQI 18dB): PING
>>> PONG
Tx (6 bytes):  TX_RESULT_SUCCESS
>>> 
```

Upon selection of PRIME or G3-PLC PHY layer, the example PLC & Go is located in the following paths of the Microchip software packages:

- in case of choosing G3-PLC PHY layer, `\thirdparty\g3\phy\atpl360\apps\phy_plc_and_go`
- in case of choosing PRIME PHY layer, `\thirdparty\prime_ng\phy\atpl360\apps\phy_plc_and_go`

The folder contains several subfolders, one for each of the supported Microchip evaluation kits. After selecting the evaluation kit to work with, it is required to select the subfolder of the IDE and open the project.

As an example, in case of using the evaluation kit PL360G55CF-EK (which uses SAMG55J19 + PL360 as platform), G3-PLC PHY layer and FCC as working band, the project to use is `samg55j19_pl360g55cf_ek` located in the Microchip G3-PLC sorfware package:

- In project properties, the object `BOARD` must be defined as `PL360G55CF_EK`
- In `conf_atpl360.h, ATPL360_WB` value must be `ATPL360_WB_FCC`

The application functions of the project are in file `phy_plc_and_go.c` that contains:

- Function `main()` of the project
- Hardware initialization and management functions
- Timing configuration

The `app_phy_ctl.c` file contains:

- PLC initialization and configuration functions
- Handlers for received messages, transmission and exception events

The `app_console.c` file contains the chat application which communicates with the PC using the serial port.

Last but not least, the "PLC Host Controller" and "PHY firmware" blocks shown in Figure 1-1 are common firmware modules of the Microchip PLC stacks. These firmware modules are described in the section Appendix A. PHY Layer.

1.1 Initialization

In the code of the example project `phy_plc_and_go`, the initialization of the PLC is performed calling the `phy_ctl_pl360_init()` function called in the `main()` function. This function configures the ATPL360 descriptor declared in `phy_ctl.c` and launches the load and initialization of the PL360 firmware binary.

1.1.1 PLC Hardware Initialization

The hardware related to the PLC communication is initialized by `atpl360_init()` function. This function requires as parameter a HAL wrapper.

- HAL wrapper contains the functions to interface with the hardware located in `sam/services/plc/pplc_if/atpl360/pplc_if.c`

```
/**
 * \brief Initialization of PL360 PHY Layer.
 *
 */
```

```
void phy_ctl_pl360_init(void)
{
    atpl360_dev_callbacks_t x_atpl360_cbs;
    atpl360_hal_wrapper_t x_atpl360_hal_wrp;

    /* Initialize G3 band static variable (ATPL360_WB defined in conf_atpl360.h) */
    suc_phy_band = ATPL360_WB;

    /* Initialize PL360 controller */
    x_atpl360_hal_wrp.plc_init = pplc_if_init;
    x_atpl360_hal_wrp.plc_reset = pplc_if_reset;
    x_atpl360_hal_wrp.plc_set_handler = pplc_if_set_handler;
    x_atpl360_hal_wrp.plc_send_boot_cmd = pplc_if_send_boot_cmd;
    x_atpl360_hal_wrp.plc_write_read_cmd = pplc_if_send_wrrd_cmd;
    x_atpl360_hal_wrp.plc_enable_int = pplc_if_enable_interrupt;
    x_atpl360_hal_wrp.plc_delay = pplc_if_delay;
    atpl360_init(&sx_atpl360_desc, &x_atpl360_hal_wrp);
```

1.1.2 Callback Setting

The example project defines three callbacks to manage different events:

- Data_indication event is managed by `_handler_data_ind()` function to process all the PLC messages received
- Data_confirm event is managed by `_handler_dat_cfm()` function to process the results of sending a PLC message
- Exception events are managed by `_handler_exception_event()` function
- Add-ons event callback is not defined because no Microchip PLC tool is used with this example

```
/**
 * \brief Initialization of PL360 PHY Layer.
 *
 */
void phy_ctl_pl360_init(void)
{
    atpl360_dev_callbacks_t x_atpl360_cbs;
[…]
    /* Callback functions configuration. Set NULL as Not used */
    x_atpl360_cbs.data_confirm = _handler_data_cfm;
    x_atpl360_cbs.data_indication = _handler_data_ind;
    x_atpl360_cbs.exception_event = _handler_exception_event;
    x_atpl360_cbs.addons_event = NULL;
    sx_atpl360_desc.set_callbacks(&x_atpl360_cbs);
```

The pointers to the callbacks are stored in the structure `x_atpl360_cbs` which is used as parameter for `set_callbacks()` function.

1.1.3 PL360 Enabling

The binary of PL360 is loaded from flash memory and its integrity is checked.

```
/**
 * \brief Initialization of PL360 PHY Layer.
 *
 */
void phy_ctl_pl360_init(void)
{
[…]
    /* Enable PL360 device: Load binary */
    _pl360_enable();
}
```

1.2 Event Handling

To check if there are any pending events from PL360 and manage them, PL360 events handler `atpl360_handle_events()` is executed every cycle of the main loop. In this example, it is called inside the function `phy_ctl_process()`.

```
/**
 * \brief Phy Controller module process.
 */
void phy_ctl_process(void)
{
    /* Manage PL360 exceptions. At initialization ATPL360_EXCEPTION_RESET is reported */
    if (sb_exception_pend) {
        /* Clear exception flag */
        sb_exception_pend = false;

        /* Set PL360 specific configuration from application */
        /* Called at initialization and if an exception occurs */
        /* If an exception occurs, PL360 is reset and some parameters may have to be 
reconfigured */
        _set_pl360_configuration();

        /* Setup G3-PLC parameters to use in transmission */
        _setup_tx_parameters();
    }

    /* Check ATPL360 pending events. It must be called from application periodically to 
handle PHY Layer events */
    atpl360_handle_events();
}
```

This function also checks if any exception related to PLC has been reported.

Note: Take into account that, during initialization, after loading the firmware binary, a reset of the PL360 is required to boot up the new firmware which also generates an exception.

1.3 Transmission

In the PLC & Go application, to transmit a message, it is only needed to call the function `phy_ctl_send_msg()` passing as parameters the pointer to the data buffer and the data length. The application includes the length of the message to transmit in the first two bytes of the data buffer so that, at the reception of the message, it is possible to know the real length of the message.

Any message transmitted by the PL360 requires a header including some transmission parameters. These parameters and the data are stored in the structure `sx_tx_msg` which is explained in detail in section 2. Transmission of PLC Messages.

The PLC & Go application only configures some transmission parameters related to the modulation. The rest of the parameters required to send a PLC message are defined during the initialization stage in the function `_setup_tx_parameters()` called after the event `ATPL360_EXCEPTION_RESET` triggered after the firmware binary load as shown in the code below:

```
static tx_msg_t sx_tx_msg;
[…]
void phy_ctl_process(void)
{
    /* Manage PL360 exceptions. At initialization ATPL360_EXCEPTION_RESET is reported */
    if (sb_exception_pend) {
        /* Clear exception flag */
        sb_exception_pend = false;

        /* Set PL360 specific configuration from application */
        /* Called at initialization and if an exception ocurrs */
        /* If an exception occurs, PL360 is reset and some parameters may have to be 
reconfigured */
        _set_pl360_configuration();

        /* Setup G3-PLC parameters to use in transmission */
        _setup_tx_parameters();
    }

    /* Check ATPL360 pending events. It must be called from application periodically to
handle PHY Layer events */
    atpl360_handle_events();
}
```

The function `_setup_tx_parameters()` configures the transmission parameters and stores the values in the structure. It also stores the maximum length of a PLC message returned by the function `_get_max_psdu_len()` taking into account the configuration of the transmission.

In case of G3-PLC, the maximum PSDU length depends on the modulation, Reed-Solomon configuration, tone mask and tone map. These parameters are set in the PL360 by means of the PIB ATPL360_REG_MAX_PSDU_LEN_PARAMS (0x4043) and then, the PL360 returns the calculated length in the PIB ATPL360_REG_MAX_PSDU_LEN (0x4042). In case of PRIME, no calculation is done by the PL360 and the values are directly defined in the function `_get_max_psdu_len()`.

Once the transmission is configured, the function `phy_ctl_send_msg()` prepares the data to be sent by PLC, storing the data in the transmission buffer (`sx_tx_msg.puc_data_buf`) and indicating the length of the data buffer (`sx_tx_msg.us_data_len`). When all the information is in the transmission structure, the function `sx_atpl360_desc.send_data(&sx_tx_msg)` sends the PLC message.

Due to G3-PLC requirements, before sending the message, the data buffer is accommodated adding some padding bytes if it is necessary. Zero bit padding is used to fit the encoded bits into a number of OFDM symbols that is a multiple of 4. In the example, the PL360 is configured to add padding and CRC automatically, enabling PIB ATPL360_REG_CRC_TX_RX_CAPABILITY (0x401C), only available for the G3-PLC firmware. In case of PRIME, no padding is required and CRC checking is implemented in the API functions.

1.4 Reception

When a PLC message is received, it is processed in the callback function `_handler_data_ind()`. This function receives a structure of type `rx_msg_t` as a parameter containing all the available data from the message. The structure is explained in detail in the section 3. Reception of PLC Messages.

The callback function checks if CRC validation was OK, extracts some data about the modulation and signal received and then, sends the data content to the application function `app_console_handle_rx_msg()` by means of the callback function `phy_ctl_rx_msg()`.

The application function `app_console_handle_rx_msg()` checks the length of the message and sends the content and information about the signal quality to the host console.

1.5 Modifications

The project example can be easily modified to be adapted to customer point-to-point application. For example:

- If the customer application requires more periodical tasks, they can be directly included in the `while()` loop of the main function.
- The communication with the PL360 requires to execute the PHY controller module process as fast as possible and it is executed in the main loop. Furthermore, a 1 ms timer is defined in `TC_1MS_Handler()` function to execute periodical tasks. In this example, it is only used to blink a led, but if the customer tasks require a timer multiple of 1 ms, new flags can be added in the same way.
- Most of the transmission parameters are set at the initialization stage but, if it is required, they can be modified dynamically. The example includes modulation changes as an example of how to do it.
- The data_indication handler is able to access all the parameters of the received message but the chat task only requires some of them. It can be easily modified to pass more or less parameters to the upper layer.

2. Transmission of PLC Messages

2.1 Transmission in G3-PLC

Transmission messages are composed using data structure `tx_msg` defined in `atpl360_comm.h`.

```
/* ! \name G3 Structure defining Tx message */
typedef struct tx_msg {
    uint8_t *puc_data_buf;                          /* Pointer to data buffer */
    uint32_t ul_tx_time;                            /* Instant when transmission has to start 
referred to 1ms PHY counter */
    uint16_t us_data_len;                           /* Length of the data buffer */
    uint8_t puc_preemphasis[NUM_SUBBANDS_MAX];      /* Preemphasis for transmission */
    uint8_t puc_tone_map[TONE_MAP_SIZE_MAX];        /* Tone Map to use on transmission */
    uint8_t uc_tx_mode;                             /* Transmission Mode */
    uint8_t uc_tx_power;                            /* Power to transmit */
    enum mod_types uc_mod_type;                     /* Modulation type */
    enum mod_schemes uc_mod_scheme;                 /* Modulation scheme */
    uint8_t uc_pdc;                                 /* Phase Detector Counter */
    uint8_t uc_2_rs_blocks;                         /* Flag to indicate whether 2 RS blocks 
have to be used (only used for FCC) */
    enum delimiter_types uc_delimiter_type;         /* DT field to be used in header */
} tx_msg_t;
```

Where:
- `puc_data_buf`: Pointer to the buffer containing the data to transmit
- `ul_tx_time`: Delay to send the message in ms referred to PL360 internal timer
- `us_data_len`: length of the data buffer containing the data to transmit
- `puc_preemphasis`: Attenuation of subbands. Not used
- `puc_tone_map`: Dynamic notching of carriers in the subband
- `uc_tx_mode`: Transmission mode

uc_tx_mode	Description
TX_MODE_ABSOLUTE	The message is sent at the specified time, referred to PL360 internal timer (1 us). Time defined in `ul_tx_time`
TX_MODE_RELATIVE	The message is sent with a delay referred to the transmission request time. Delay defined in `ul_tx_time`
TX_MODE_FORCED	Transmission has a higher priority than a reception in progress
TX_MODE_SYNCP_CONTINUOUS	Continuous transmission of the preamble. Used for testing
TX_MODE_SYMBOLS_CONTINUOUS	Continuous transmission of a message. Used for testing
TX_MODE_CANCEL	Cancels the ongoing transmission (started or programmed)

- `uc_tx_power`: Signal attenuation (3dBs/unit, and 0 means maximum signal level)
- `uc_mod_type`: Modulation type

uc_mod_type	Value	Description
MOD_TYPE_BPSK	0	BPSK Modulation
MOD_TYPE_QPSK	1	QPSK Modulation
MOD_TYPE_8PSK	2	8PSK Modulation
MOD_TYPE_BPSK_ROBO	4	BPSK Robust Modulation

- `uc_mod_scheme`: Modulation scheme

uc_mod_scheme	Value	Description
MOD_SCHEME_DIFFERENTIAL	0	Modulation Scheme Differential
MOD_SCHEME_COHERENT	1	Modulation Scheme Coherent

- `uc_pdc`: Phase detector counter
- `uc_2_rs_blocks`: Number of Reed-Solomon blocks used (only valid for FCC band).
- `uc_delimiter_type`: Delimiter type used in the header

uc_delimiter_type	Value	Description
DT_SOF_NO_RESP	0	Acknowledgment is not requested
DT_SOF_RESP	1	Acknowledgment is requested
DT_ACK	2	Positive acknowledgement
DT_NACK	3	Negative acknowledgement

Once the desired data is in the transmission buffer and the rest of the transmission parameters are completed, the message can be transmitted using the function `send_data` defined in the atpl360 descriptor (see section atpl360_init() function).

Before sending the data, G3-PLC PHY layer requires to accommodate the data to the requirements of G3-PLC including a padding to fit the encoded bits into a number of OFDM symbols and adding a CRC value. The PL360 firmware binary allows to enable the calculation of the padding and the CRC in the PL360, this functionality is managed by the PIB ATPL360_REG_CRC_TX_RX_CAPABILITY (0x401C) and it is disabled by default but enabled in the PLC & Go application.

After sending the message to be transmitted to the PL360 device, a TX data confirm event is triggered indicating the result of the transmission. This event is managed by the `data_confirm` callback configured by `set_callbacks()` function. This callback receives as parameter a data structure of type `tx_cfm_t` which is defined in `atpl360_comm.h`.

```
/* ! \name G3 Structure defining result of a transmission */
typedef struct tx_cfm {
    /* RMS_CALC it allows to estimate tx power injected */
    uint32_t ul_rms_calc;
    /* Instant when frame transmission ended referred to 1ms PHY counter */
    uint32_t ul_tx_time;
    /* Tx Result (see "TX Result values" above) */
    enum tx_result_values uc_tx_result;
} tx_cfm_t;
```

The event returns one of the following transmission result values:

```
/* ! \name G3 TX Result values */
enum tx_result_values {
    TX_RESULT_SUCCESS = 1,       /* ended successfully */
    TX_RESULT_INV_LENGTH = 2,    /* invalid length error */
    TX_RESULT_BUSY_CH = 3,       /* busy channel error */
    TX_RESULT_BUSY_TX = 4,       /* busy in transmission error */
    TX_RESULT_BUSY_RX = 5,       /* busy in reception error */
    TX_RESULT_INV_SCHEME = 6,    /* invalid modulation scheme error */
    TX_RESULT_TIMEOUT = 7,       /* timeout error */
    TX_RESULT_INV_TONEMAP = 8,   /* invalid tone map error */
    TX_RESULT_INV_MODTYPE = 9,   /* Invalid modulation type error */
    TX_RESULT_INV_DT = 10,       /* Invalid delimiter type */
    TX_RESULT_NO_TX = 255,       /* No transmission ongoing */
};
```

2.2 Transmission in PRIME

Transmission messages are composed using data structure `tx_msg` defined in `atpl360_comm.h`.

```
/* ! \name PRIME Structure defining Tx message */
typedef struct tx_msg {
    uint32_t ul_tx_time;            // Time for transmission in us
    uint16_t us_data_len;           // Length of the data buffer
    uint8_t uc_att_level;           // Attenuation level with which the message must be
transmitted
    enum mod_schemes uc_scheme;     // Modulation scheme of last transmitted message
    uint8_t uc_disable_rx;          // TX Forced
    enum mode_types uc_mod_type;    // Type A, Type B or Type BC
    uint8_t uc_tx_mode;             // Transmission Mode
    enum buffer_id uc_buffer_id;    // Buffer Id used for transmission
    uint8_t uc_rsvd;                // Reserved byte
    uint8_t *puc_data_buf;          // Pointer to data buffer
} tx_msg_t;
```

Where:

- `ul_tx_time`: Delay to send the message in us. Depending on the transmission mode (`uc_tx_mode`), it is a relative or absolute time
- `us_data_len`: Length of the data to be sent
- `uc_att_level`: Set transmission attenuation power (1 dB per unit, 0 = maximum signal level)
- `uc_scheme`: Modulation scheme of the transmitted message

uc_scheme	Value	Description
MOD_SCHEME_DBPSK	0	Differential BPSK Modulation
MOD_SCHEME_DQPSK	1	Differential QPSK Modulation
MOD_SCHEME_D8PSK	2	Differential 8PSK Modulation
MOD_SCHEME_DBPSK_C	4	Differential BPSK Modulation + Convolutional code
MOD_SCHEME_DQPSK_C	5	Differential QPSK Modulation + Convolutional code
MOD_SCHEME_D8PSK_C	6	Differential 8PSK Modulation + Convolutional code
MOD_SCHEME_R_DBPSK	12	Differential BPSK Robust Modulation
MOD_SCHEME_R_DQPSK	13	Differential BPSK Robust Modulation

- `uc_disable_rx`: When it is enabled, transmission has a higher priority than a reception in progress
- `uc_mod_type`: Modulation type of the message (introduced in PRIME 1.4 for backward compatibility with PRIME 1.3)

uc_mod_type	Value	Description
MODE_TYPE_A	0	Frame type A
MODE_TYPE_B	2	Frame type B
MODE_TYPE_BC	3	Frame type BC

- `uc_tx_mode`: Transmission mode

uc_tx_mode	Description
TX_MODE_ABSOLUTE	The message is sent at the specified time, referred to PL360 internal timer (1 us). Time defined in `ul_tx_time`

...........continued

uc_tx_mode	Description
TX_MODE_RELATIVE	The message is sent with a delay referred to the transmission request time. Delay defined in `ul_tx_time`
TX_MODE_CANCEL	Cancels the ongoing transmission (started or programmed)
TX_MODE_PREAMBLE_CONTINUOUS	Continuous transmission of the preamble. Used for testing
TX_MODE_SYMBOLS_CONTINUOUS	Continuous transmission of a message. Used for testing

- `uc_buffer_id`: Buffer to store the transmission data. There are two available buffers. But transmissions cannot be overlapped in time. In that case, TX_BUFFER_0 has priority

uc_buffer_id	Description
TX_BUFFER_0	First transmission buffer
TX_BUFFER_1	Second transmission buffer

- `uc_rsvd`: Reserved
- `puc_data_buf`: Pointer to the buffer containing the data to transmit. The size of the data is defined in `us_data_len`

Once the desired data is in the transmission buffer and the rest of the transmission parameters are completed, the message can be transmitted using the function `send_data` defined in the atpl360 descriptor (see section atpl360_init() function).

After sending the message to be transmitted to the PL360 device, a TX data confirm event is triggered indicating the result of the transmission. This event is managed by the `data_confirm` callback configured by `set_callbacks()` function. This callback receives as parameter a data structure of type `tx_cfm_t` which is defined in `atpl360_comm.h`.

```
/* ! \name Structure defining result of a transmission */
typedef struct tx_cfm {
    /** Transmission time in us. */
    uint32_t ul_tx_time;
    /** RMS value emitted */
    uint32_t ul_rms_calc;
    /** Type mode: Type A, Type B or Type BC  */
    enum mode_types uc_mod_type;
    /** TX Result */
    enum tx_result_values uc_tx_result;
    /** Buffer Id used for transmission to confirm */
    enum buffer_id uc_buffer_id;
} tx_cfm_t;
```

The event returns one of the following transmission result values:

```
/* ! \name TX Result values */
enum tx_result_values {
    TX_RESULT_PROCESS = 0,        /* Transmission result: already in process */
    TX_RESULT_SUCCESS = 1,        /* Transmission result: ended successfully */
    TX_RESULT_INV_LENGTH = 2,     /* Transmission result: invalid length error */
    TX_RESULT_BUSY_CH = 3,        /* Transmission result: busy channel error */
    TX_RESULT_BUSY_TX = 4,        /* Transmission result: busy in transmission error */
    TX_RESULT_BUSY_RX = 5,        /* Transmission result: busy in reception error */
    TX_RESULT_INV_SCHEME = 6,     /* Transmission result: invalid modulation scheme error */
    TX_RESULT_TIMEOUT = 7,        /* Transmission result: timeout error */
    TX_RESULT_INV_BUFFER = 8,     /* Transmission result: invalid buffer identifier error */
    TX_RESULT_INV_MODE = 9,       /* Transmission result: invalid Prime Mode error */
    TX_RESULT_INV_TX_MODE = 10,   /* Transmission result: invalid transmission mode */
    TX_RESULT_CANCELLED = 11,     /* Transmission result: Transmission cancelled */
    TX_RESULT_NO_TX = 255,        /* Transmission result: No transmission ongoing */
```

3. Reception of PLC Messages

3.1 Reception in G3-PLC

When PL360 receives a message, an event of RX data indication is triggered. This event is managed by the data_indication callback `_handler_data_ind`. This callback receives as parameter a data structure of type `rx_msg_t`, which is defined in `atpl360_comm.h`.

```
/* ! \name G3 Structure defining Rx message */
typedef struct rx_msg {
    uint32_t ul_rx_time;
    uint32_t ul_frame_duration;
    uint16_t us_rssi;
    uint16_t us_data_len;
    uint8_t uc_zct_diff;
    uint8_t uc_rs_corrected_errors;
    enum mod_types uc_mod_type;
    enum mod_schemes uc_mod_scheme;
    uint32_t ul_agc_factor;
    uint16_t us_agc_fine;
    int16_t ss_agc_offset_meas;
    uint8_t uc_agc_active;
    uint8_t uc_agc_pga_value;
    int16_t ss_snr_fch;
    int16_t ss_snr_pay
    uint16_t us_payload_corrupted_carriers;
    uint16_t us_payload_noised_symbols;
    uint8_t uc_payload_snr_worst_carrier;
    uint8_t uc_payload_snr_worst_symbol;
    uint8_t uc_payload_snr_impulsive;
    uint8_t uc_payload_snr_band;
    uint8_t uc_payload_snr_background;
    uint8_t uc_lqi;
    enum delimiter_types uc_delimiter_type;
    uint8_t uc_crc_ok;
    uint8_t puc_tone_map[TONE_MAP_SIZE_MAX];
    uint8_t puc_carrier_snr[PROTOCOL_CARRIERS_MAX];
    uint8_t *puc_data_buf;
} rx_msg_t;
```

The structure contains all the information available about the message received. The fields of the structure are:

`ul_rx_time`	Instant when frame was received (end of frame) referred to 1µs PHY counter
`ul_frame_duration`	Frame duration in µs (Preamble + FCH + Payload)
`us_rssi`	Received Signal Strength Indicator in dBµV
`us_data_len`	Length of received frame in bytes
`uc_zct_diff`	Phase difference with transmitting node in multiples of 60 degrees
`uc_rs_corrected_errors`	Number of errors corrected by Reed-Solomon
`uc_mod_type`	Modulation type of the last received frame. Related constants defined in section 2.1 Transmission in G3-PLC
`uc_mod_scheme`	Modulation scheme of the last received frame. Related Constants defined in section 2.1 Transmission in G3-PLC
`ul_agc_factor`	Global amplifying factor of the main branch (21 bits)
`us_agc_fine`	Factor that multiplies the digital input signal (13 bits)
`ss_agc_offset_meas`	DC offset after the ADC that will be removed in case the DC Blocker is enabled (10 bits)

`uc_agc_active`	Flag to indicate if AGC is active
`uc_agc_pga_value`	Gain value applied to the PGA (3 bits)
`ss_snr_fch`	SNR of the header in quarters of dBs
`ss_snr_pay`	SNR of the payload in quarters of dBs
`us_payload_corrupted_carriers`	Number of corrupted carriers in payload due to narrow/broad-band noise
`us_payload_noised_symbols`	Number of corrupted symbols in payload due to impulsive noise
`uc_payload_snr_worst_carrier`	SNR for the worst case carrier of the payload in quarters of dBs
`uc_payload_snr_worst_symbol`	SNR for the worst case symbol of the payload in quarters of dBs
`uc_payload_snr_impulsive`	SNR of corrupted symbols in payload due to impulsive noise in quarters of dBs
`uc_payload_snr_band`	SNR of corrupted carriers in payload due to narrow/broad-band noise in quarters of dBs
`uc_payload_snr_background`	SNR without taking into account corrupted carriers and symbols in quarters of dBs
`uc_lqi`	Link Quality Indicator. SNR in quarters of dBs with offset of 10 dB (value 0 means -10 dB)
`uc_delimiter_type`	DT field coming in header. Related Constants defined in section 2.1 Transmission in G3-PLC
`uc_crc_ok`	CRC verification result (1: OK; 0: BAD; 0xFE: unexpected error; 0xFF: CRC capability disabled. See PIB ATPL360_REG_CRC_TX_RX_CAPABILITY (0x401C))
`puc_tone_map`	Tone Map in received frame. Related constants explained below
`puc_carrier_snr`	SNR for each carrier in dBs (with offset of 10dB, i.e. value 0 means -10dB). Related constants explained below
`puc_data_buf`	Pointer to data buffer containing received frame. The received frame includes padding (if needed). CRC is included if the CRC capability in the PL360 is disabled

Related symbolic constants affecting `puc_tone_map` parameter:

```
/* ! Subbands for Cenelec-A bandplan */
    #define NUM_SUBBANDS_CENELEC_A 6
/* ! Subbands for FCC bandplan */
    #define NUM_SUBBANDS_FCC 24
/* ! Subbands for ARIB bandplan */
    #define NUM_SUBBANDS_ARIB 18
/* ! Subbands for Cenelec-B bandplan */
    #define NUM_SUBBANDS_CENELEC_B 4
/* ! Tone Map size for Cenelec bandplan */
    #define TONE_MAP_SIZE_CENELEC 1
/* ! Tone Map size for FCC and ARIB bandplans */
    #define TONE_MAP_SIZE_FCC_ARIB 3
/* ! Maximum number of tone map */
    #define TONE_MAP_SIZE_MAX TONE_MAP_SIZE_FCC_ARIB
/* ! Maximum number of subbands */
    #define NUM_SUBBANDS_MAX NUM_SUBBANDS_FCC
```

Related constants affecting `puc_carrier_snr` parameter:

```
/* ! Carriers for CENELEC-A bandplan */
    #define NUM_CARRIERS_CENELEC_A      36
/* ! Carriers for FCC bandplan */
    #define NUM_CARRIERS_FCC            72
/* ! Carriers for ARIB bandplan */
```

```
    #define NUM_CARRIERS_ARIB           54
/* ! Carriers for CENELEC-B bandplan */
    #define NUM_CARRIERS_CENELEC_B      16
/* ! Maximum number of protocol carriers */
    #define PROTOCOL_CARRIERS_MAX       NUM_CARRIERS_FCC
```

3.2 Reception in PRIME

When PL360 receives a message, an event of RX data indication is triggered. This event is managed by the ATPL360 device callback `data_indication` assigned to `_handler_data_ind` during the initialization of the example. This callback receives as parameter a data structure of type `rx_msg_t` which is defined in `atpl360_comm.h`.

```
/* ! \name Structure defining Rx message */
typedef struct rx_msg {
    uint32_t ul_evm_header_acum;
    uint32_t ul_evm_payload_acum;
    uint32_t ul_rx_time;
    uint16_t us_evm_header;
    uint16_t us_evm_payload;
    uint16_t us_data_len;
    enum mod_schemes uc_scheme;
    enum mode_types uc_mod_type;
    enum header_types uc_header_type;
    uint8_t uc_rssi_avg;
    uint8_t uc_cinr_avg;
    uint8_t uc_cinr_min;
    uint8_t uc_ber_soft;
    uint8_t uc_ber_soft_max;
    uint8_t uc_nar_bnd_percent;
    uint8_t uc_imp_percent;
    uint8_t *puc_data_buf;
} rx_msg_t;
```

Where:

`ul_evm_header_acum`	Accumulated Error Vector Magnitude for header
`ul_evm_payload_acum`	Accumulated Error Vector Magnitude for payload
`ul_rx_time`	Reception time (start of message) referred to 1us PHY counter
`us_evm_header`	Error Vector Magnitude for header
`us_evm_payload`	Error Vector Magnitude for payload
`us_data_len`	Length of the data buffer in bytes
`uc_scheme`	Modulation scheme of the received message. Related constants defined in section 2.2 Transmission in PRIME
`uc_mod_type`	Type A, Type B or Type BC frames. Related constants defined in section 2.2 Transmission in PRIME
`uc_header_type`	Header Type of the received message
`uc_rssi_avg`	Average RSSI (Received Signal Strength Indication)
`uc_cinr_avg`	Average CNIR (Carrier to Interference + Noise ratio)
`uc_cinr_min`	Minimum CNIR (Carrier to Interference + Noise ratio)
`uc_ber_soft`	Average Soft BER (Bit Error Rate)
`uc_ber_soft_max`	Maximum Soft BER (Bit Error Rate)
`uc_nar_bnd_percent`	Percentage of carriers affected by narrow band noise
`uc_imp_percent`	Percentage of symbols affected by impulsive noise

`*puc_data_buf`	Pointer to local data buffer

Regarding uc_header_type, the PRIME specification defines different types of MAC PDUs for different purposes in upper layers. In this application, only Generic MAC PDU is used.

uc_header_type	Description
PHY_HT_GENERIC	Generic PDU are used for all data traffic and most control traffic
PHY_HT_PROMOTION	Promotion Needed PDU sent by disconnected nodes
PHY_HT_BEACON	Beacon PDU sent by switch devices.

4. PHY Controller

The PHY controller is a C library (`app_phy_ctl.c` and `app_phy_ctl.h`) included in the PLC & Go example that defines a simple API for a point-to-point or point-to-multipoint application (in the case of the example included, a chat application) simplifying the use of PLC communications with PL360. The API is composed of a series of functions and callbacks to manage the events coming from the PL360 device.

4.1 G3-PLC PHY Controller

API Callbacks

The callback functions are organized in a structure:

```
/* Phy Controller API callbacks */
typedef struct phy_ctl_callbacks {
    phy_data_confirm_t phy_ctl_data_confirm;
    phy_data_indication_t phy_ctl_rx_msg;
    phy_data_indication_bad_crc_t phy_ctl_rx_msg_discarded;
    phy_update_tx_cfg_t phy_ctl_update_tx_configuration;
} phy_ctl_callbacks_t;
```

The `phy_ctl_data_confirm` callback is executed to manage the event generated after the transmission of a message sent by the PL360 to notify the result, which is passed as a parameter (See list of events in 2.1 Transmission in G3-PLC).

The `phy_ctl_rx_msg` callback is executed when a message is received and it passes the checking of the CRC. The main parameters passed by the callback are the pointer to the data buffer and the length of the data. Other parameters about signal quality are passed (modulation scheme and type, RSSI and LQI) as example.

The `phy_ctl_rx_msg_discarded` callback is executed when a corrupted message is received.

The `phy_ctl_update_tx_configuration` callback is executed when the transmission parameters are configured. Current modulation scheme, modulation type and maximum data length for a frame are sent as parameters.

API Functions

The `phy_ctl_pl360_init` function loads the binary file with the firmware of the PL360 and initializes the callbacks required by PL360 Host Controller. For G3-PLC, It requires as parameter an identifier of the initial frequency working band:

```
/*! \name ATPL360 work band identifiers
 */
/* ! @{ */
/* ! CENELEC A Band Plan (35 - 91 Khz) */
#define ATPL360_WB_CENELEC_A                       1
/* ! FCC Band Plan (154 - 488 Khz) */
#define ATPL360_WB_FCC                             2
/* ! ARIB Band Plan (154 - 404 Khz) */
#define ATPL360_WB_ARIB                            3
/* ! CENELEC-B Band Plan (98 - 122 Khz) */
#define ATPL360_WB_CENELEC_B                       4
/* ! @} */
```

The `phy_ctl_set_callbacks` function initializes the callbacks required to evaluate the different events coming from the PLC. The callbacks are listed in the struct `phy_ctl_callbacks`.

The `phy_ctl_process` function manages the pending events coming from PLC, this function has to be called periodically in the main loop of the application.

The `phy_ctl_send_msg` function sends a message through PLC. This function requires two parameters:

- A pointer to the buffer which contains the message
- Length of the message in bytes

This function returns the status of the message:

```
    TX_RESULT_PROCESS = 0,                   /* Transmission result: already in process */
    TX_RESULT_INV_LENGTH = 2,                /* Transmission result: invalid length error */
    TX_RESULT_NO_TX = 255,                   /* Transmission result: No transmission ongoing */
```

The returned value is not the result of the PLC transmission, it is just the transmission of the message from the host to the PL360. In case of transmission in progress, the status of the PLC transmission (TX_RESULT_PROCESS) is notified by the callback `phy_ctl_Data_confirm`.

The `phy_ctl_get_mod_scheme` returns the modulation scheme used in the transmission.

The `phy_ctl_set_mod_scheme` changes the modulation scheme configured to be used in the transmission.

The `phy_ctl_get_mod_type` returns the modulation type used in the transmission.

The `phy_ctl_set_mod_type` changes the modulation type configured to be used in the transmission.

The `phy_ctl_get_band` returns the identifier for the selected working band for the PLC transmission.

The `phy_ctl_set_band` changes the selected working band for the PLC transmission. The identifier of the new frequency working band is passed as a parameter. When the working band changes, the PL360 transceiver is reset to load the appropriate firmware (only for G3-PLC Multiband).

4.2 PRIME PHY Controller

API Callbacks

The callbacks functions are organized in a structure:

```
/* Phy Controller API callbacks */
typedef struct phy_ctl_callbacks {
    phy_data_confirm_t phy_ctl_data_confirm;
    phy_data_indication_t phy_ctl_rx_msg_crc_ok;
    phy_data_indication_bad_crc_t phy_ctl_rx_msg_bad_crc;
    phy_update_tx_cfg_t phy_ctl_update_tx_configuration;
} phy_ctl_callbacks_t;
```

The `phy_ctl_data_confirm` callback is executed after the event of the transmission of a message, it is sent by the PL360 to notify the result. The result of the transmission is passed as a parameter (see list of events in 2.2 Transmission in PRIME).

The `phy_ctl_rx_msg_crc_ok` callback is executed when a message is received and it passes the checking of the CRC (calculated by the host device). The main parameters passed by the callback are the pointer to the data buffer and the length of the data. Other parameters about signal quality are passed (modulation scheme, RSSI and average CINR).

The `phy_ctl_rx_msg_bad_crc` callback is executed when a corrupted message is received.

The `phy_ctl_update_tx_configuration` callback is executed when the transmission parameters are configured. Current Modulation Scheme and maximum data length for a frame are sent as parameters.

API Functions

The `phy_ctl_pl360_init` function loads the binary file with the firmware of the PL360 and initializes the callbacks required by PL360 Host Controller. It requires as parameter the number of the initial frequency working channel (from 1 to 8).

The `phy_ctl_set_callbacks` function initializes the callbacks required to evaluate the different events coming from the PLC. The callbacks are listed in the struct `phy_ctl_callbacks`.

The `phy_ctl_process` function manages the pending events coming from PLC, this function has to be called periodically in the main loop of the application.

The `phy_ctl_send_msg` function sends a message through PLC. This function requires two parameters:

- A pointer to the buffer which contains the message

- Length of the message in bytes

This function returns the status of the message:

```
    TX_RESULT_PROCESS = 0,                      /* Transmission result: already in process */
    TX_RESULT_INV_LENGTH = 2,                   /* Transmission result: invalid length error */
    TX_RESULT_NO_TX = 255,                      /* Transmission result: No transmission ongoing */
```

The returned value is not the result of the PLC transmission, it is just the transmission of the message from the host to the PL360. In case of transmission in progress, the status of the PLC transmission (TX_RESULT_PROCESS) is notified by the callback `phy_ctl_Data_confirm`.

The `phy_ctl_get_mod_scheme` returns the modulation scheme used in the transmission.

The `phy_ctl_set_mod_scheme` changes the modulation scheme configured to be used in the transmission.

The `phy_ctl_get_channel` returns the number of the selected working channel for the PLC transmission.

The `phy_ctl_set_channel` changes the selected working channel for the PLC transmission. The number of the new channel is passed as a parameter.

5. Appendix A. PHY Layer

The PHY layer is in charge of frame transmission and reception. The PL360 device runs the PHY layer firmware, thus data indication and data confirm events are triggered to upper layers by the host controller module. Additionally to these events, the PHY layer API (through the host controller module) implements entry functions in order to transmit a frame using the PLC modem, to perform periodic tasks and to access the PHY Information Base (PIB) to read or write parameters.

To get deeper information about the PHY layer API and how to configure and use it, please check the document *50002738 PL360 Host Controller* and the stack user guides (*50002728 G3-PLC Firmware stack User Guide* for G3-PLC or *50002759 PRIME 1.4 Firmware stack for Service Node User Guide* for PRIME).

5.1 PL360 Configuration

A PLC communication project based on PL360 requires to apply an initial configuration controlled by several PIBs (Physical Information Base).

Microchip PLC stacks use two ways to configure these parameters:

- Symbolic constants. These are usually hardware related and not modified by the user after compiling the project. For example: default G3-PLC working band (ATPL360_WB) defines a set of parameters which are dependent on the frequency band used to communicate via PLC (defined in file `conf_atpl360.h`).

Symbolic Constant	Values	Description
ATPL360_WB	ATPL360_WB_CENELEC_A	PL360 working in CENELEC-A Band
	ATPL360_WB_CENELEC_B	PL360 working in CENELEC-B Band
	ATPL360_WB_FCC	PL360 working in FCC Band
	ATPL360_WB_ARIB	PL360 working in ARIB Band

- Management primitives. These can be dynamically modified. For example, the impedance detection (ATPL360_REG_CFG_AUTODETECT_IMPEDANCE) is a functionality that can be enabled/disabled. These primitives can be modified through the PL360 Host Controller block of the stack.

5.2 PL360 Host Controller

To use the PL360 PHY layer, it is required to configure the PL360 Host Controller API in the host microcontroller which will be the interface between the user application and the PL360.

This section explains the different steps required during the initialization phase of the system. After powering up the PL360 device, a set of initialization sequences must be executed in the correct order for the proper operation of the PL360 device. The steps are the following:

1. Initialize controller descriptor: The PL360 Host Controller is initialized by calling the `atpl360_init()` function.
2. Set controller callbacks: The PL360 Host Controller reports PLC events using callback functions. There are 4 callback functions:
   – Data indication: reports a new incoming message
   – Data confirm: reports the result of the last transmitted message
   – Add-on event: reports that a new add-on (PHY tester or PHY sniffer) message is ready to be sent to the PLC application (for example, an embedded sniffer frame received event)
   – Exception event: reports if an exception occurs, such as a reset of the PL360 device
3. Enable controller: The PL360 Host Controller is enabled by calling the `atpl360_enable()` function in the API.
4. PL360 event handling: The PL360 device interrupts the host MCU when one or more events are pending in the PL360 embedded firmware. The host MCU application processes received data and events when the

PL360 Host Controller calls the corresponding event callback function(s). In order to receive event callbacks, the host MCU application is required to periodically call the `atpl360_handle_events()` function in the API.

5.2.1 atpl360_init() function

This function initializes the hardware parameters and configures the controller descriptor. This function performs the following actions:

- Sets the handlers for the PLC interruption
- Initializes the PLC SPI service
- And, if necessary, initializes add-on interfaces

The controller descriptor provides a set of functions to access the PL360 device. It is defined in a structure of function pointers as follows:

```
/* ATPL360 descriptor */
typedef struct atpl360_descriptor {
    pf_set_callbacks_t set_callbacks;
    pf_send_data_t send_data;
    pf_mng_get_cfg_t get_config;
    pf_mng_set_cfg_t set_config;
    pf_addons_event_t send_addons_cmd;
} atpl360_descriptor_t;
```

where:

- `set_callbacks` function is used to set upper layers functions to be executed when a PL360 Host Controller event has been reported
- `send_data` function provides a mechanism to send a PLC message through the PL360 device
- `get_config` function provides a read access method to get PL360 internal data (the list of the available PIBs is defined in `atpl360_reg_id`, in `atpl360_comm.h`)
- `set_config` function provides a write access method to set PL360 internal data (the list of the available PIBs is defined in `atpl360_reg_id`, in `atpl360_comm.h`)
- `send_addons_cmd` function provides a mechanism to connect PLC Microchip tools to the PL360 device. All information received from these tools should be redirected to this function to pass the information to the PL360 Host Controller

The PL360 Host Controller also needs to have access to hardware peripherals. A HAL wrapper structure is used to define this hardware and software dependency.

```
/* ATPL360 Hardware Wrapper */
typedef void (*pf_plc_init_t)(void);
typedef void (*pf_plc_reset_t)(void);
typedef void (*pf_plc_set_handler_t)(void (*p_handler)(void));
typedef bool (*pf_plc_bootloader_cmd_t)(uint16_t us_cmd, uint32_t ul_addr, uint32_t
ul_data_len, uint8_t *puc_data_buf, uint8_t *puc_data_read);
typedef bool (*pf_plc_write_read_cmd_t)(uint8_t uc_cmd, void *px_spi_data, void
*px_spi_status_info);
typedef void (*pf_plc_enable_int_t)(bool enable);
typedef void (*pf_plc_delay_t)(uint8_t uc_tref, uint32_t ul_delay);

typedef struct atpl360_hal_wrapper {
    pf_plc_init_t plc_init;
    pf_plc_reset_t plc_reset;
    pf_plc_set_handler_t plc_set_handler;
    pf_plc_bootloader_cmd_t plc_send_boot_cmd;
    pf_plc_write_read_cmd_t plc_write_read_cmd;
    pf_plc_enable_int_t plc_enable_int;
    pf_plc_delay_t plc_delay;
} atpl360_hal_wrapper_t;
```

5.2.2 setcallbacks() function

This function sets the callbacks for the different PL360 Host Controller events.

The structure used as input of pf_set_callbacks_t function contains the pointers to the functions to be executed for the different PL360 Host Controller events:

```
/* ATPL360 device Callbacks */
typedef struct atpl360_dev_callbacks {
    /* Callback for TX Data Confirm */
    pf_data_confirm_t data_confirm;
    /* Callback for RX Data Indication */
    pf_data_indication_t data_indication;
    /* Callback for Serial Data Event (see addons) */
    pf_addons_event_t addons_event;
    /* Callback for Exceptions triggered by the component */
    pf_exeption_event_t exception_event;
} atpl360_dev_callbacks_t;
```

where:

- `data_confirm` function is used to notify of the result of the last message transmission
- `data_indication` function is used to notify of the reception of a new message
- `addons_event` function is used to notify that there is a new message to be sent to a PLC Microchip Tool
- `exception_event` function is used to notify of any exception which occurs in the communication with the PL360 device

5.2.3 atpl360_enable() function

The `atpl360_enable()` function transfers the firmware binary file from Flash memory to the PL360 device and starts running it. It requires the following parameters:

Parameter	Description
ul_binary_address	Memory address location of the PL360 firmware binary
ul_binary_len	Size of the PL360 firmware binary

This function performs the following actions:

- Disable PLC interrupt
- Transfer firmware binary file to the PL360 device
- Check firmware integrity
- Enable PLC interrupt

The function returns 0 in case of success loading the PL360 binary.

5.2.4 atpl360_handle_events() function

This function provides a mechanism to notify user application about the PL360 Host Controller events by means of the callbacks previously configured with `setcallbacks()` function.

It is recommended to call this function either:

- From the main loop or from a dedicated task in the host MCU application, or
- At least once when the host MCU application receives an interrupt from the PL360 embedded firmware

This function checks all PLC events:

- PHY parameters and configuration
- End of transmission of PLC message
- End of reception of PLC message
- Exceptions

After checking the PLC events, it triggers the corresponding PL360 Host Controller callbacks.

6. Appendix B. Porting Platform

The example runs in the supported platforms specified in the Release Notes of the Microchip G3-PLC and PRIME stacks. If it is required to port the code to a new non-supported platform, the most important considerations to bear in mind are:

- Flash memory: PL360 firmware binary is stored in flash memory in the host device. The new platform must have enough free flash memory to store the binary.
- SPI bus: Communication with PL360 is made by SPI bus. The PL360 Host Controller uses two different SPI modes (8-bit mode and 16-bit mode), both of them must be supported in the new platform. Recommended bus speed is 12 MHz for FCC band and 8 MHz for CENELEC-A and CENELEC-B bands.
- Minimum pinout for PL360-host communication: The communication between PL360 and host device requires:
  - SPI: Communication bus between PL360 and host device
  - Reset line
  - External Interrupt: Line to notify of pending events in the PL360 to be processed. In the host device, an interrupt associated to this line is required
  - Carrier Detect: Only for PRIME stack
  - TST line: Only in case of using low-power modes
- Porting hardware-related source code files. The main files to be considered are located in the folders:
  - `/sam/services/plc/pplc_if/atpl360/`
  - `/common/components/plc/atpl360/`
- Memory: It depends on the platform to port the code. Take into account that hardware-related source code files in the SAMG55 require more than 4 KB of ROM and more than 3 KB in RAM (for more detailed numbers, please consult map output file in a built example).

7. Revision History

7.1 Rev A - 02/2020

Document	First issue.

The Microchip Website

Microchip provides online support via our website at http://www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to http://www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: http://www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-5625-4

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit http://www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/support Web Address: http://www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4450-2828 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820