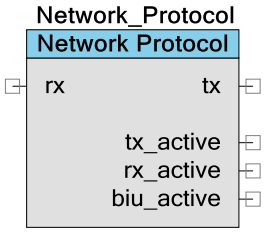
Network Protocol (Network_Protocol)

1.0

Features



- Transmits and receives data packets at 600 - 2400 bps
- 8-bit CRC insertion and verification to detect packet errors
- FEC Encoding to detect and correct single-bit errors
- Acknowledgments to verify successful transmission
- Multiple addressing modes: 8-bit (logical), 16-bit (extended logical), 64-bit (physical), or group
- Band-in-use (BIU) detection to avoid packet collisions
- Frequency hopping to automatically switch to the FSK center frequency of the incoming signal
- Compatible with the FSK Modulator and FSK Demodulator components, as well as the PSoC® 1 powerline communication solution

General Description

The Network Protocol performs bi-directional half-duplex communication (either the transmitter or receiver is active, but not both) over a bus with a single master or multiple masters (a master can initiate communication) and multiple slaves. The Network Protocol performs two primary functions: Cypress Packet Protocol processing and Receive Signal Analysis.

The Cypress Packet Protocol generates and verifies data packets that are optimized for transmission over noisy transmission media. Each packet contains an 8-bit CRC that is used by the receiver to verify that the packet is error-free. If that packet is error-free and the master expects an acknowledgment, then the slave will send back an acknowledgment so that the master knows that the transmission was a success. If the master does not receive an acknowledgment, it will re-transmit the original packet.

The Receive Signal Analysis function processes signal information from the FSK Demodulator to perform band-in-use (BIU) detection and frequency hopping. BIU detection is required for networks that can have two systems attempting to transmit at the same time. In this situation, the Network Protocol will first check if there is a signal present in the communication band (known as band-in-use). If it detects a BIU condition, the Network Protocol will wait for the BIU condition to stop before it transmits. This minimizes the chance of a data packet collision on the network. When frequency hopping is enabled, the Network Protocol will automatically switch the FSK

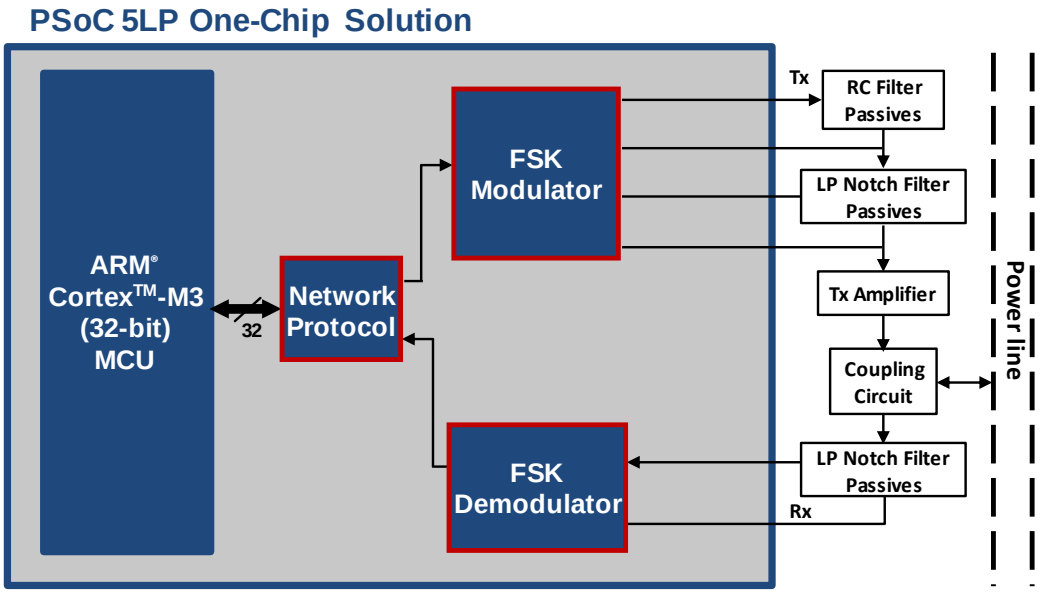
Demodulator's FSK center frequency to match the center frequency of the received signal. The host application is responsible for selecting the FSK Modulator's center frequency when transmitting a new packet. For example, the host application may want to switch FSK center frequencies if the success rate on the current frequency is less than 90%. When the master system changes frequencies and transmits data packets, the slave system will automatically switch to the new frequency and receive the data packets.

When to Use a Network Protocol

The purpose of a network protocol is to ensure that the correct data is sent to the intended destination, in a timely manner. This component is also designed for communication over a bus with one or more masters and slaves. Since there can be multiple masters, it is also important to prevent data collisions.

It is most commonly used for powerline communication (PLC). Figure 1 shows a block diagram of a PLC system. The packet of data is created and serialized by the Network Protocol component. The FSK Modulator then converts the serial signal into an FSK signal, which can be coupled to the powerline by a coupling circuit (described in application note AN76458 "PSoC 5LP High Voltage (120 – 240 VAC) Powerline Communication"). The signal is received on the other end by the receiver, which uses the FSK Demodulator component to convert the FSK signal back into a serial signal for the Network Protocol.

Figure 1. Powerline Communication System Diagram



Input/Output Connections

This section describes the various input and output connections for the Network Protocol. An asterisk (*) in the list of I/Os states that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

rx – Input*

The rx terminal is the serial data input. It is present when the Cypress_Packet_Protocol parameter is set to Enabled.

pclk – Input*

The pclk terminal is the external clock input for the Network Protocol's system timer. The frequency (or nearest integer) of this clock should match the Protocol_Clock_External_Frequency parameter. It is present when the Protocol_Clock_Source parameter is set to External.

uclk – Input*

The uclk terminal is the external clock input for the Network Protocol's UART. The baud rate of the UART will be 1/8th of the frequency of this clock. It is present when the UART_Clock_Source parameter is set to External.

tx – Output*

The tx terminal is the serial data output. It is present when the Cypress_Packet_Protocol parameter is set to Enabled.

tx_active – Output *

The tx_active terminal is the transmit active indicator. When the protocol is transmitting data, the output is high. Otherwise, it is low. It is present when the Status_Indicators parameter is set to Enabled.

rx_active – Output *

The rx_active terminal is the receive active indicator. When the protocol receives a byte of data that matches the Cypress packet structure, this output goes high. Up until the destination address is received, this output pulses high for less than 20 us after each received byte. If the destination address of the received packet matches the local address, this output stays on until the device receives the last byte or an invalid byte. It is present when the Status_Indicators parameter is set to Enabled.

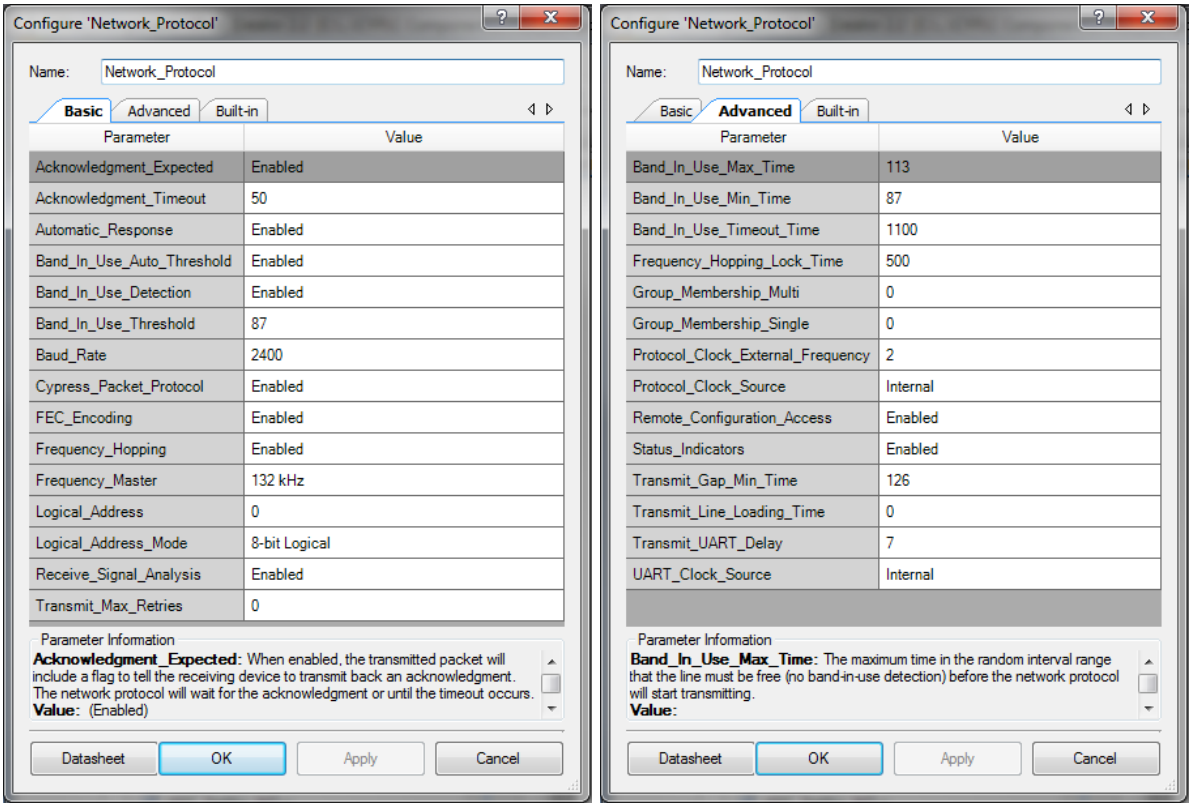
biu_active – Output *

The biu_active terminal is the band-in-use (BIU) active indicator. When the network protocol is checking if the line is busy, this output goes high when there is a signal above the threshold that is set by the Band_In_Use_Threshold parameter. It is present when the Status_Indicators parameter is set to Enabled.

Component Parameters

Drag a Network Protocol onto your design and double click it to open the Configure dialog. **Error! Reference source not found.** shows the Configure dialog.

Figure 2. Configure Network Protocol Dialog



The Network Protocol provides the following parameters.

Basic Options

Acknowledgment Expected

When this parameter is Enabled, the Network Protocol sets the ACK MODE flag to ‘1’ in the data packet header of transmitted packets. If a device receives this packet, it will transmit back an acknowledgment packet to confirm reception of the data packet. The Network Protocol of the device that transmitted the data packet will wait until the acknowledgment is received or the Acknowledgment_Timeout time elapses. If the acknowledgment is received, it sets the TX_COMPLETE flag in the return value of the Network_Protocol_ReadTxStatus function. If the timeout occurs, the Network Protocol will re-transmit the packet until an acknowledgment is received or it reaches the limit set by the Transmit_Max_Retries parameter. If it reaches the limit, it sets the ACK_TIMEOUT flag in the return value of the Network_Protocol_ReadTxStatus function.

When this parameter is set to Disabled, the Network Protocol does not set the ACK MODE flag in the transmitted packet and it does not wait for an acknowledgment. When it finishes transmitting the packet, it sets the TX_COMPLETE flag in the return value of the Network_Protocol_ReadTxStatus function.

This parameter is ignored if the Cypress_Packet_Protocol parameter is Disabled. The default is Enabled.

Acknowledgment Timeout

This parameter defines the amount of time (in milliseconds) that the network protocol waits to receive an acknowledgment packet after it transmits a data packet. It is ignored if the Acknowledgments Expected parameter is Disabled or if the Cypress_Packet_Protocol parameter is Disabled. The default is 50.

Automatic_Response

When this parameter is Enabled, the Network Protocol to automatically send back a response when it receives a data packet with the ACK MODE flag equal to ‘1’ or when the Command ID of the received data packet is one of the configuration commands (see the COMMAND ID section for the list of configuration commands) . This parameter is ignored if the Cypress_Packet_Protocol parameter is Disabled. The default is Enabled.

Band_In_Use_Auto_Threshold

When this parameter is Enabled, the Network Protocol automatically sets the BIU detection threshold based on the noise on the line. When it is Disabled, the Network Protocol uses the Band_In_Use_Threshold parameter as the detection threshold. This parameter is ignored if the Band_In_Use_Detection parameter is Disabled or if Receive_Signal_Analysis is Disabled. The default is Enabled.

Band_In_Use_Detection

When this parameter is Enabled, the Network Protocol checks if there is a signal present in the FSK communication frequency (band-in-use) before transmitting. This minimizes the chance of a packet collision from two devices transmitting at the same time. The Network Protocol waits for the line to be free (no band-in-use detection) for a random interval between the range set by the Band_In_Use_Min_Time and Band_In_Use_Max_Time parameters. If the line is free for this interval, the Network Protocol will transmit the packet. If there is a band-in-use detection, the network protocol will select a new random interval and wait for the line to be free for that amount of time. If the total wait time is longer than the Band_In_Use_Timeout_Time parameter, the Network Protocol will stop checking the line and will set the BIU_TIMEOUT flag in the return value of the Network_Protocol_ReadTxStatus function. The default is Enabled.

When this parameter is disabled, the band-in-use detection code will not be generated in the Network Protocol source files. Therefore, if you plan to use band-in-use detection during run time, then set this parameter to Enabled. It can be enabled or disabled with the Network_Protocol_EnableBiuDetection and Network_Protocol_DisableBiuDetection APIs, respectively.

Band_In_Use_Threshold

This parameter defines the threshold for the band-in-use detector. If the Network Protocol is checking if the line is free and there is a signal on the line that exceeds this threshold, then it is a band-in-use detection event. This parameter is ignored if the Band_In_Use_Detection parameter is Disabled. The default is 87.

Baud_Rate

This parameter defines the baud rate of the transmitted data and of the expected received data. The range is 600 – 2400 bps. This parameter is ignored if the Cypress_Packet_Protocol parameter is Disabled. The default is 2400.

Cypress_Packet_Protocol

When this parameter is Enabled, the Network Protocol supports the transmission and reception of data packets. When it is Disabled, only the signal processing functions (band-in-use detection, frequency hopping) are supported. The default is Enabled.

FEC_Encoding

When this parameter is Enabled, the Network Protocol will encode each 4-bit nibble of each byte of a transmitted packet into an 8-bit byte. The most-significant nibble is encoded and transmitted first. For each transmitted byte, if there is a single-bit error in the byte, the receiver will detect the error and correct it. The receiver will decode the 8-bit byte back into the original 4-bit nibble. This parameter is ignored if the Cypress_Packet_Protocol parameter is Disabled. The default for this parameter is Disabled.

When this parameter is disabled, the FEC code will not be generated in the Network Protocol source files. Therefore, if you plan to use FEC encoding during run time, then set this parameter to Enabled. It can be enabled or disabled with the Network_Protocol_EnableFec and Network_Protocol_DisableFec APIs, respectively.

Frequency_Hopping

When this parameter is Enabled, the Network Protocol will automatically switch the FSK Center Frequency of the FSK Demodulator and FSK Modulator so that it matches the center frequency of the received signal. If the Network Protocol receives a valid Cypress packet, it will stay locked to the center frequency for the time defined by the Frequency_Hopping_Lock_Time parameter.

If the Network Protocol receives a packet that has the ACK MODE flag set to '1', then it will transmit back an acknowledgment that is modulated with the same FSK Center Frequency as the received data packet.

In order to track the received frequency, the Transmit_UART_Delay parameter of the transmitter's Network Protocol must be at least 14. The Frequency_Hopping parameter is ignored if Receive_Signal_Analysis is Disabled. The default is Disabled.

When this parameter is disabled, the frequency hopping code will not be generated in the Network Protocol source files. Therefore, if you plan to use frequency hopping during run time, then set this parameter to Enabled. It can be enabled or disabled with the Network_Protocol_EnableFrequencyHopping and Network_Protocol_DisableFrequencyHopping APIs, respectively.

Frequency_Master

This parameter defines the FSK Center Frequency that the Network Protocol is processing signal information from. It should match the FSK_Center_Frequency parameters of the FSK Modulator and FSK Demodulator components. The default is 132 kHz.

Logical_Address

This parameter defines the logical address of this Network Protocol. When the Network Protocol receives a packet that has the destination type set to logical, it only accepts the packet if the destination address matches the logical address. When the Network Protocol transmits a packet with the source address type set to logical, the packet's source address field will contain the logical address. This parameter is ignored if the Cypress_Packet_Protocol parameter is Disabled. The default is 0.

Logical_Address_Mode

This parameter defines the logical address mode of this Network Protocol. When it is 8-bit Logical, the logical address is 8 bits. When it is 16-bit Extended, the logical address is 16 bits. The Network Protocol changes the size of the packet to accommodate the Logical Address Mode. This parameter must be the same for both the transmitting device and receiving device if

a packet is transmitted with a logical destination address type. This parameter is ignored if the Cypress_Packet_Protocol parameter is Disabled. The default is 8-bit Logical.

Receive_Signal_Analysis

When this parameter is Enabled, the Network Protocol will process the measured peak signal from the FSK Demodulator to determine the noise floor and the maximum signal on the line. This information is used for frequency hopping and for band-in-use detection. The default is Enabled.

Transmit_Max_Retries

This parameter defines the maximum number of times that the Network Protocol will re-transmit the data packet if it does not receive an acknowledgment packet. This parameter is ignored if the Cypress_Packet_Protocol parameter is Disabled. The default is 0.

Advanced Options

Band_In_Use_Max_Time

This parameter defines the maximum time (in milliseconds) of the band-in-use detection interval range. Before transmitting a packet, the Network Protocol randomly chooses a value within this range and then checks if the line is free for this entire interval. This parameter is ignored if the Band_In_Use_Detection parameter is Disabled. The default is 113.

Band_In_Use_Min_Time

This parameter defines the minimum time (in milliseconds) of the band-in-use detection interval range. Before transmitting a packet, the Network Protocol randomly chooses a value within this range and then checks if the line is free for this entire interval. This parameter is ignored if the Band_In_Use_Detection parameter is Disabled. The default is 87.

Band_In_Use_Timeout_Time

This parameter defines the total amount of time (in milliseconds) that the Network Protocol can check if the line is free before stopping and notifying the application. This parameter is ignored if the Band_In_Use_Detection parameter is Disabled. The default is 1100.

Frequency_Hopping_Lock_Time

This parameter defines the total amount of time (in milliseconds) that the Network Protocol will stay locked to the FSK center frequency if it receives a valid packet. This parameter is ignored if the Frequency_Hopping parameter is Disabled. The default is 500.

Group_Membership_Multi

This parameter defines the multiple groups that the device belongs to. This 8-bit field is one-hot encoded (for example, 0b00100001 means that the device belongs to groups 6 and 1). If the Network Protocol receives a packet with the group destination addressing type and the group address matches one of the enabled groups, then the protocol will store the packet. This parameter is ignored if the Cypress_Packet_Protocol parameter is Disabled. The default is 0.

Group_Membership_Single

This parameter defines the single 8-bit group that the device belongs to. If the Network Protocol receives a packet with the group destination addressing type and the group address matches this group, then the protocol will store the packet. This parameter is ignored if the Cypress_Packet_Protocol parameter is Disabled. The default is 0.

Protocol_Clock_External_Frequency

This parameter defines the frequency (in kHz) of the external protocol clock source (pclk). If the frequency of the clock source is not an integer, then enter the closest integer value. This is ignored when The Protocol_Clock_Source is Internal. The default is 2.

To maintain compliance with CENELEC for the band-in-use timing, the user should account for the error between the actual clock frequency and the value of this parameter in the Band_In_Use_Max_Time, Band_In_Use_Min_Time, Band_In_Use_Timeout_Time, and Transmit_Gap_Min_Time parameters. For example, if the actual clock frequency is 2.4 kHz and this value is 2, the actual times are 20% shorter than expected. Therefore, the user should increase these parameters by a factor of 20%. Additionally, all of the Network Protocol timing variables are 16-bit unsigned integers. The code calculates the variable's value by multiplying the parameter value by this Protocol_Clock_External_Frequency. Therefore, the user must make sure that the calculated value is not greater than 65535 for any of the parameters. By default, the longest parameter time is the Band_In_Use_Timeout_Time (1100). Therefore, the maximum Protocol_Clock_External_Frequency is 59 when using the default values.

Protocol_Clock_Source

This parameter defines the source of the protocol clock (pclk). When it is Internal, the Network Protocol uses a new 2 kHz digital clock. When it is External, the user supplies an external clock. When External, the user should enter the frequency of the external clock in the Protocol_Clock_External_Frequency parameter. The default is Internal.

Remote_Configuration_Access

When this parameter is Enabled, a configuration command (for example, CMD_SET_LOGICAL_ADDRESS) will be able to change the Network Protocol's configuration. When Disabled, the configuration will not change and the Network Protocol will respond with a denied request. This parameter is ignored if the Cypress_Packet_Protocol parameter is Disabled or the Automatic_Response parameter is disabled. The default is Enabled.

Status_Indicators

When this parameter is Enabled, the three output terminals tx_active, rx_active, and biu_active are present. These terminals represent when the Network Protocol is transmitting, in the middle of receiving a packet, or has detected a band-in-use event, respectively. The default is Enabled.

Transmit_Gap_Min_Time

This parameter defines the minimum amount of time to wait for the line to be free after transmitting a packet. It ensures that other devices on the line get a chance to transmit next. It should be larger than the Band_In_Use_Max_Time parameter. This parameter is ignored if the Band_In_Use_Detection parameter is Disabled. The default is 126 ms, which is compliant with CENELEC.

Transmit_Line_Loading_Time

This parameter defines the amount of time (in milliseconds) that the Network Protocol will turn on the external transmit amplifier before it starts the FSK Modulator. This is useful when the receiver on the other end of the line wants to measure the noise when the line is loaded by the transmitter. This helps to calibrate the Frequency Hopping algorithm. This parameter is ignored if the Cypress_Packet_Protocol parameter is Disabled. The default is 0.

Transmit_UART_Delay

This parameter defines the amount of time (in milliseconds) that the Network Protocol will turn on the FSK Modulator before it starts transmitting data. This is needed for the receiver on the other end of the line to clear its buffer and be aligned to the first bit of transmitted data. It should be at least 2 ms longer than the time between bytes of data. When transmitting to a device that has Frequency Hopping enabled, this time should be at least 14. This parameter is ignored if the Cypress_Packet_Protocol parameter is Disabled. The default is 7.

UART_Clock_Source

This parameter defines the source of the internal UART clock (uclk). When it is Internal, the Network Protocol uses a new digital clock based on the value of the Baud_Rate parameter. When it is External, the user supplies an external clock that should be eight times the value of the Baud_Rate parameter. The default is Internal.

Clock Selection

The clock tolerance for the UART Clock and Protocol Clock should be a maximum of ±2 percent.

Resources

The Network Protocol uses the following device resources. The number of resources depends on the value of the Cypress_Packet_Protocol parameter.

Configuration				Digital Blocks					
Cypress_Packet_Protocol	Status_Indicators	UART_Clock_Source	Protocol_Clock_Source	Datapath Cells	Macro cells	Status Cells	Control Cells	Timer Blocks	Digital Clocks
Enabled	Enabled	Internal	Internal	3	22	2	2	1	2
Enabled	Disabled	Internal	Internal	3	22	2	1	1	2
Enabled	Enabled	External	Internal	3	22	2	1	1	1
Enabled	Enabled	Internal	External	3	22	2	1	1	1
Disabled	Disabled	N/A	External	0	1	0	0	1	0

API Memory Usage

The component memory usage varies significantly, depending on the number of APIs used and component configuration. The following table provides the memory usage for all APIs available in the given component configuration.

The measurements were performed with the associated compiler configured in Release mode and with optimization set for Size. For a specific design, the map file generated by the compiler can be analyzed to determine the memory usage.

Configuration							API Memory (Bytes)	
Cypress_Packet_Protocol	Automatic_Response	FEC_Encoding	Band_In_Use_Detection	Receive_Signal_Analysis	Frequency_Hopping	Status_Indicators	Flash	RAM
Enabled	Enabled	Enabled	Enabled	Enabled	Enabled	Enabled	8464	352
Disabled	N/A	N/A	Enabled	Enabled	Enabled	Enabled	2464	112
Enabled	Disabled	Enabled	Enabled	Enabled	Enabled	Enabled	6840	352
Enabled	Enabled	Disabled	Enabled	Enabled	Enabled	Enabled	7992	344
Enabled	Enabled	Enabled	Disabled	Enabled	Enabled	Enabled	7520	360
Enabled	Enabled	Enabled	Enabled	Disabled	N/A	Enabled	6808	352
Enabled	Enabled	Enabled	Enabled	Enabled	Disabled	Enabled	7624	352
Enabled	Enabled	Enabled	Enabled	Enabled	Enabled	Disabled	8200	352
Disabled	N/A	N/A	Disabled	Disabled	N/A	Disabled	16	0

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "Network_Protocol_1" to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "Network_Protocol".

Packet-Related Functions (Network_Protocol_Packet.c)

Function	Description
Network_Protocol_Start()	Starts the protocol state machine, UART, and timer
Network_Protocol_Init()	Initializes the default configuration provided by the customizer
Network_Protocol_Stop()	Stops the UART and the timer.
Network_Protocol_Poll()	Runs the main protocol state machine, which sequences the transmission and reception of data, and band-in-use detection to avoid collisions.
Network_Protocol_SetLogicalAddressMode()	Sets the logical addressing mode when transmitting and receiving packets.
Network_Protocol_SetLogicalAddress()	Sets the logical address of this Network Protocol.
Network_Protocol_GetPhysicalAddress()	Gets the 64-bit physical address from the device's manufacturing ID
Network_Protocol_AddToGroup()	Adds membership to a group.
Network_Protocol_RemoveFromGroup()	Removes membership from a group.
Network_Protocol_SetSourceAddressType()	Sets the source address type of the transmitted packets.
Network_Protocol_SetDestinationAddress()	Sets the destination address type and destination address of the transmitted packets.
Network_Protocol_LoadTxData()	Loads the command ID and data payload and sets a flag to start transmitting the packet when the line is free.
Network_Protocol_ReadTxStatus()	Returns the current status of the transmitter.
Network_Protocol_ReadRxStatus()	Returns the current status of the receiver.
Network_Protocol_GetRxDestinationAddressType()	Returns the destination address type of the received packet.
Network_Protocol_GetRxSourceAddressType()	Returns the source address type of the received packet.
Network_Protocol_GetRxSourceAddress()	Returns the source address of the received packet.

Network_Protocol_GetRxCommandID()	Returns the type of command in the received packet.
Network_Protocol_GetRxDataLength()	Returns the length of the payload in the received packet.
Network_Protocol_GetRxData()	Returns the payload data of the received packet.
Network_Protocol_EnableAck()	Enables acknowledgments to be expected by the Network Protocol after it transmits a packet.
Network_Protocol_DisableTxAck()	Acknowledgments will not be expected by the Network Protocol after it transmits a packet.
Network_Protocol_SetTxRetries()	Sets the maximum number of times that the Network Protocol will re-transmit the data packet if it does not receive an acknowledgment packet.
Network_Protocol_EnableConfigAccess()	Allows any special configuration command to change the configuration of this Network Protocol.
Network_Protocol_DisableConfigAccess()	Prevents a special configuration command from changing the configuration of this Network Protocol.
Network_Protocol_EnableFec()	Enables FEC encoding of the transmitted packets.
Network_Protocol_DisableFec()	Disables FEC encoding of the transmitted packets.
Network_Protocol_SetTestMode()	Sets the test mode of the Network Protocol
Network_Protocol_SetBaudRate()	Sets the baud rate of the internal UART that transmits and receives packets
Network_Protocol_SetUartDelayTime()	Sets the amount of time from when the FSK Modulator is started to when the UART starts transmitting data.
Network_Protocol_EnableTxLineLoading()	Enables the Network Protocol to turn on the external amplification circuit to load the line for the specified amount of time before it starts the FSK Modulator to transmit the FSK signal
Network_Protocol_DisableTxLineLoading()	Disables line loading so that the Network Protocol turns on the amplification circuit at the same time that it starts the FSK Modulator to transmit the FSK signal.

Signal-Related Functions (Network_Protocol_Signal.c)

Function	Description
Network_Protocol_EnableBiuDetection()	Enables the band-in-use detection algorithm, which checks if there is a signal present in the FSK communication frequency (band-in-use) before transmitting.
Network_Protocol_DisableBiuDetection()	Disables the band-in-use detection algorithm.
Network_Protocol_SetBiuThreshold()	Sets the threshold for band-in-use detection.
Network_Protocol_EnableAutoBiuThreshold()	Enables the algorithm that sets the BIU threshold to be above the noise floor of the line.

Network_Protocol_DisableAutoBiuThreshold()	Disables the algorithm that sets the BIU threshold to be above the noise floor of the line.
Network_Protocol_SetBiuWaitInterval()	Sets the minimum and maximum time (in milliseconds) of the band-in-use detection interval range.
Network_Protocol_SetBiuTimeout()	Sets the total amount of time (in milliseconds) that the Network Protocol can check if the line is free.
Network_Protocol_EnableFrequencyHopping()	Enables the Network Protocol to automatically switch the FSK Demodulator's center frequency to match the frequency of the received FSK signal.
Network_Protocol_DisableFrequencyHopping()	Disables frequency hopping. The Network Protocol will not change the FSK center frequency.
Network_Protocol_SetMasterFrequency()	Sets the active FSK center frequency that is being analyzed. It should match the FSK center frequency of the FSK Demodulator.
Network_Protocol_GetActiveFrequency()	Returns the FSK center frequency that is being analyzed.
Network_Protocol_GetNewFskLevel()	Returns the latest reading of the peak amplitude (in dB) of the filtered received signal.
Network_Protocol_GetMaxFskLevel()	Returns the maximum peak amplitude over the interval specified by the Network_Protocol_SetMaxSignalTiming API.
Network_Protocol_GetNoiseFloor()	Returns the noise floor (in dB) of the line.
Network_Protocol_SetMaxSignalTiming()	Sets the amount of time (in milliseconds) that the maximum amplitude is stored for.
Network_Protocol_SetNoiseFloorTiming()	Sets the timing for measuring the noise floor.
Network_Protocol_InitializeSignalAnalysis()	Initializes the receive signal analysis algorithm and data structure.
Network_Protocol_ResetSignalAnalysis()	Resets the timer that checks if the filter has stabilized so that the proper amplitude is measured.
Network_Protocol_RunSignalAnalysis()	Collects the statistics (noise floor and maximum) of the filtered received signal.
Network_Protocol_InitializeBiuDetection()	Initializes the band-in-use detection variables.
Network_Protocol_StartAcquirePowerline()	Starts the algorithm to check if the line is free.
Network_Protocol_GetLineStatus()	Checks if the line has been free for the random band-in-use (BIU) interval
Network_Protocol_PauseAcquirePowerline()	Pauses the acquire powerline timers and carrier detection.
Network_Protocol_ResumeAcquirePowerline()	Restarts the acquire powerline timers and carrier detection.
Network_Protocol_StartTxWait()	Starts the timer that tracks the time since the last transmission.
Network_Protocol_TestFrequencyHop()	Tests if the receiver should switch to the other FSK frequency based on the latest FSK amplitude reading and the noise floor.
Network_Protocol_RunFrequencyHop()	Switches the FSK Demodulator to the other FSK frequency.

Network_Protocol_StartTimer()	Starts the 16-bit system timer.
Network_Protocol_StopTimer()	Stops the 16-bit system timer.
Network_Protocol_ReadTimer()	Returns the current 16-bit value of the system timer.

Global Variables

Function	Description
Network_Protocol_initVar (static)	The initVar variable indicates the initial configuration of this component. This variable is prepended with the component name. The variable is initialized to zero and set to 1 the first time Network_Protocol_Start() is called. This allows for component initialization without reinitialization in all subsequent calls to the Network_Protocol_Start() routine.

void Network_Protocol_Start(void)

Description: This is the preferred method to begin component operation. This function starts the protocol state machine, UART, and timer. On the first call of this function, it sets the Network_Protocol_initVar variable and calls the Network_Protocol_Init() function.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_Init(void)

Description: Initializes or restores the component according to the customizer Configure dialog settings. It is not necessary to call Network_Protocol_Init() because the Network_Protocol_Start() API calls this function and is the preferred method to begin component operation.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_Stop(void)

Description: Stops the UART and the timer.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_Poll(void)

Description: Runs the main protocol state machine, which sequences the transmission and reception of data, and band-in-use detection to avoid collisions.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_SetLogicalAddressMode(uint8 addressMode)

Description: Sets the logical addressing mode when transmitting and receiving packets. When a packet is transmitted with logical addresses, the source address and destination address will either contain an 8-bit address or a 16-bit address. When a packet is received, the Network Protocol will either look for an 8-bit address or a 16-bit address.

Parameters: uint8 addressMode: one of the following addressing modes

Options	Description
`$INSTANCE_NAME`_NORMAL_LOGICAL_MODE	An 8-bit logical address is used
`$INSTANCE_NAME`_EXTENDED_LOGICAL_MODE	A 16-bit logical address is used

Return Value: None

Side Effects: None

void Network_Protocol_SetLogicalAddress(uint8 address)

Description: Sets the logical address of this Network Protocol. When a packet with a logical destination address type is received, its destination address is compared with this address. When the Network Protocol transmits a packet with logical source address type, the packet will contain this address as the source address.

Parameters: uint8 addressMode: one of the following addressing modes

Options	Description
`$INSTANCE_NAME`_NORMAL_LOGICAL_MODE	An 8-bit logical address is used
`$INSTANCE_NAME`_EXTENDED_LOGICAL_MODE	A 16-bit logical address is used

Return Value: None

Side Effects: None

uint8 Network_Protocol_GetPhysicalAddress (uint8 *physicalAddress)

Description: Gets the 64-bit physical address from the device's manufacturing ID

Parameters: physicalAddress: a pointer to where the 64-bit address will be stored

Return Value: None

Side Effects: None

void Network_Protocol_AddToGroup(uint8 groupNumber)

Description: Adds membership to a group. When a packet is received that has the group destination addressing type (Network_Protocol_DA_TYPE_GROUP) and the destination address is a group that this device is a member of, then the Network Protocol will store the packet and notify the host. Otherwise, it ignores it. The device can belong to up to 9 groups at a time. It can belong to all of groups 1-8 plus one more group in the range of 9-255.

Parameters: groupNumber: the group that this device will be a member of. The valid range is 1-255.

Return Value: None

Side Effects: None

void Network_Protocol_RemoveFromGroup(uint8 groupNumber)

Description: Removes membership from a group. The device can belong to up to 9 groups at a time. It can belong to all of groups 1-8 plus one more group in the range of 9-255.

Parameters: groupNumber: the group that this device will not be a member of. The valid range is 1-255.

Return Value: None

Side Effects: None

void Network_Protocol_ SetSourceAddressType(uint8 addressType)

Description: Sets the source address type of the transmitted packets.

Parameters: addressType: the source address type (logical or physical).

Options	Description
Network_Protocol_SA_TYPE_LOGICAL	Logical address
Network_Protocol_SA_TYPE_PHYSICAL	Physical address

Return Value: None

Side Effects: None

void Network_Protocol_SetDestinationAddress(uint8 addressType, uint8 * destinationAddress)

Description: Sets the destination address type and destination address of the transmitted packets

Parameters: addressType: the destination address type (logical, physical, or group) The values are:

Options	Description
Network_Protocol_DA_TYPE_LOGICAL	Logical Address
Network_Protocol_DA_TYPE_GROUP	Group Address
Network_Protocol_DA_TYPE_PHYSICAL	Physical Address

destinationAddress: a pointer to the destination address

Return Value: None

Side Effects: None

void Network_Protocol_LoadTxData (uint8 commandID, uint8 *payload, uint8 payloadLength)

Description: Loads the command ID and data payload and sets a flag to start transmitting the packet when the line is free.

Parameters: commandID: the type of command

Options	Description
Network_Protocol_CMD_SET_TX_ENABLE	Enable (payload = 1) or disable (payload = 0) the transmitter
Network_Protocol_CMD_SET_EXT_MODE	Set logical addressing mode to 16-bit (payload = 1) or normal (payload = 0)
Network_Protocol_CMD_SET_LOGICAL_ADDRESS	Set the logical address to the value in payload
Network_Protocol_CMD_GET_LOGICAL_ADDRESS	Request the logical address.
Network_Protocol_CMD_GET_PHYSICAL_ADDRESS	Request the physical address
Network_Protocol_CMD_GET_PLC_MODE	(Legacy Support with PSoC 1) Request the PLC_MODE register contents
Network_Protocol_CMD_GET_FW_VERSION	Request the firmware version
Network_Protocol_CMD_DATA	Transmit the data in the payload
Network_Protocol_CMD_REQUEST_DATA	Send the payload along with a request for a response. Wait for the response for up to 1.5 s.
Network_Protocol_CMD_RESPONSE_DATA	Send the payload as a response to the Network_Protocol_CMD_REQUEST_DATA command
Network_Protocol_CMD_SET_BIU_DISABLE	Disable (payload = 1) or enable (payload = 0) band-in-use detection
Network_Protocol_CMD_SET_BIU_THRESHOLD	Set the band-in-use threshold to the value of the payload
Network_Protocol_CMD_SET_GROUPS	Set the single group member (first payload byte) and the multi-group member (second byte)
Network_Protocol_CMD_GET_GROUPS	Request the single group membership (first response byte) and multi-group membership (second byte)

User-defined command value (>= 16)	Transmit the data in the payload

payload: a pointer to the data payload
payloadLength: the length of the data payload

Return Value: None

Side Effects: None

uint8 Network_Protocol_ReadTxStatus(void)

Description: Returns the current status of the transmitter. The status is automatically cleared.

Parameters: None

Return Value: Current status of the transmitter.

Options	Description
Network_Protocol_TX_COMPLETE	The transmission is complete and if acknowledgments are enabled, the acknowledgment was received
Network_Protocol_TX_IN_PROGRESS	The transmission is in progress
Network_Protocol_ACK_WAITING	The transmission is complete and the Network Protocol is waiting for an acknowledgment
Network_Protocol_RESPONSE_TIMEOUT	The transmission of a request packet is complete and the Network Protocol is waiting for a response
Network_Protocol_ACK_TIMEOUT	The transmission is complete, but the acknowledgment wasn't received within the timeout window
Network_Protocol_BIU_TIMEOUT	The line has been busy for the BIU timeout window and the Network Protocol has stopped checking if the line is free
Network_Protocol_BIU_IN_PROGRESS	The Network Protocol is checking if the line is free. It has not started transmitting yet.

Side Effects: None

uint8 Network_Protocol_ReadRxStatus(void)

Description: Returns the current status of the receiver. The status is automatically cleared.

Parameters: None

Return Value: Current status of the receiver.

Options	Description
Network_Protocol_RX_IN_PROGRESS	The receiver has received at least one byte of a new packet
Network_Protocol_RX_COMPLETE	The receiver has received a new packet
Network_Protocol_RX_OVERFLOW	The receiver has an unread packet in its buffer and had to drop a new packet

Side Effects: None

uint8 Network_Protocol_GetRxDestinationAddressType (void)

Description: Returns the destination address type of the received packet.

Parameters: None

Return Value: Destination address type.

Options	Description
Network_Protocol_DA_TYPE_LOGICAL	Logical Address
Network_Protocol_DA_TYPE_GROUP	Group Address
Network_Protocol_DA_TYPE_PHYSICAL	Physical Address

Side Effects: None

uint8 Network_Protocol_GetRxSourceAddressType (void)

Description: Returns the source address type of the received packet.

Parameters: None

Return Value: Source address type.

Options	Description
Network_Protocol_SA_TYPE_LOGICAL	Logical address
Network_Protocol_SA_TYPE_PHYSICAL	Physical address

Side Effects: None

void Network_Protocol_GetRxSourceAddress(uint8 * sourceAddress)

Description: Gets the source address of the received packet.

Parameters: sourceAddress: A pointer to the source address that this function will write to.

Return Value: None

Side Effects: None

uint8 Network_Protocol_GetRxCommandID (void)

Description: Returns the command ID of the received packet.

Parameters: None

Return Value: The command ID.

Options	Description
Network_Protocol_CMD_SET_TX_ENABLE	Enable (payload = 1) or disable (payload = 0) the transmitter
Network_Protocol_CMD_SET_EXT_MODE	Set logical addressing mode to 16-bit (payload = 1) or normal (payload = 0)
Network_Protocol_CMD_SET_LOGICAL_ADDRESS	Set the logical address to the value in payload
Network_Protocol_CMD_GET_LOGICAL_ADDRESS	Request the logical address.
Network_Protocol_CMD_GET_PHYSICAL_ADDRESS	Request the physical address
Network_Protocol_CMD_GET_PLC_MODE	(Legacy Support with PSoC 1) Request the PLC_MODE register contents
Network_Protocol_CMD_GET_FW_VERSION	Request the firmware version
Network_Protocol_CMD_DATA	Transmit the data in the payload
Network_Protocol_CMD_REQUEST_DATA	Send the payload along with a request for a response. Wait for the response for up to 1.5 s.
Network_Protocol_CMD_RESPONSE_DATA	Send the payload as a response to the Network_Protocol_CMD_REQUEST_DATA command
Network_Protocol_CMD_SET_BIU_DISABLE	Disable (payload = 1) or enable (payload = 0) band-in-use detection
Network_Protocol_CMD_SET_BIU_THRESHOLD	Set the band-in-use threshold to the value of the payload
Network_Protocol_CMD_SET_GROUPS	Set the single group member (first payload byte) and the multi-group member (second byte)
Network_Protocol_CMD_GET_GROUPS	Request the single group membership (first response byte) and multi-group

	membership (second byte)
User-defined command value (>= 16)	Transmit the data in the payload

Side Effects: None

uint8 Network_Protocol_GetRxDataLength (void)

Description: Returns the source address type of the received packet.

Parameters: None

Return Value: The length of the payload in the received packet.

Side Effects: None

void Network_Protocol_GetRxData(uint8 * rxData)

Description: Gets the payload data of the received packet.

Parameters: rxData: A pointer to the variable that this function will store the data in

Return Value: None

Side Effects: None

void Network_Protocol_EnableAck (uint16 ackWaitTime)

Description: Enables acknowledgments to be expected by the Network Protocol after it transmits a packet. The Network Protocol sets the ACK MODE flag to '1' in the data packet header of transmitted packets. If a device receives this packet, it will transmit back an acknowledgment packet to confirm reception of the data packet. The Network Protocol of the device that transmitted the data packet will wait until the acknowledgment is received or the ackTimeoutTime time elapses. If the acknowledgment is received, it sets the Network_Protocol_TX_COMPLETE flag in the return value of the Network_Protocol_ReadTxStatus function. If the timeout occurs, the Network Protocol will re-transmit the packet until an acknowledgment is received or it reaches the limit set by the retryCount variable. If it reaches the limit, it sets the Network_Protocol_ACK_TIMEOUT flag in the return value of the Network_Protocol_ReadTxStatus function.

Parameters: ackTimeoutTime: the amount of time (in milliseconds) that the network protocol waits to receive an acknowledgment packet after it transmits a data packet

Return Value: None

Side Effects: None

void Network_Protocol_DisableAck (void)

Description: Acknowledgments will not be expected by the Network Protocol after it transmits a packet. The Network Protocol does not set the ACK MODE flag in the transmitted packet and it does not wait for an acknowledgment. When it finishes transmitting the packet, it sets the Network_Protocol_TX_COMPLETE flag in the return value of the Network_Protocol_ReadTxStatus function.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_SetTxRetries (uint8 numRetries)

Description: Acknowledgments will not be expected by the Network Protocol after it transmits a packet. The Network Protocol does not set the ACK MODE flag in the transmitted packet and it does not wait for an acknowledgment. When it finishes transmitting the packet, it sets the Network_Protocol_TX_COMPLETE flag in the return value of the Network_Protocol_ReadTxStatus function.

Parameters: numRetries: the maximum number of retries.

Return Value: None

Side Effects: None

void Network_Protocol_EnableConfigAccess (void)

Description: Allows any special configuration command to change the configuration of this Network Protocol. The configuration command IDs are the commands that start with Network_Protocol_CMD_SET

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_DisableConfigAccess (void)

Description: Acknowledgments will not be expected by the Network Protocol after it transmits a packet. The Network Protocol does not set the ACK MODE flag in the transmitted packet and it does not wait for an acknowledgment. When it finishes transmitting the packet, it sets the Network_Protocol_TX_COMPLETE flag in the return value of the Network_Protocol_ReadTxStatus function.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_EnableFec (void)

Description: Enables FEC encoding of the transmitted packets. Each byte (including the preamble and CRC) is FEC encoded. Each 4-bit nibble (most significant first) is encoded into an 8-bit byte. If there is a single bit error in this byte during transmission, the receiver's FEC decoder will be able to detect the error and correct it.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_DisableFec (void)

Description: Disables FEC encoding of the transmitted packets.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_SetTestMode(uint8 testMode)

Description: Sets the test mode of the Network Protocol

Parameters: uint8 testMode: one of the following addressing modes

Options	Description
Network_Protocol_TEST_MODE_IGNORE_ADDRESS	The network protocol will ignore the destination address of the received packet and will store the packet if the CRC is correct. It will not transmit an acknowledgment.
Network_Protocol_TEST_MODE_IGNORE_CRC	The network protocol will ignore any CRC errors in the received packet. It will store the packet if the destination address matches its local address.

Return Value: None

Side Effects: None

void Network_Protocol_SetBaudRate(uint16 baudRate)

Description: Sets the baud rate of the internal UART that transmits and receives packets

Parameters: baudRate: the baud rate (in bits per second)

Return Value: None

Side Effects: None

void Network_Protocol_SetUartDelayTime(uint16 delayTime)

Description: Sets the amount of time from when the FSK Modulator is started to when the UART starts transmitting data. This delay gives the receiver's UART time to clear its buffer and prepare to receive the start bit of the next transmitted packet. It should be longer than the period from the start of one byte to the start of the next byte. At 2400 bps, that period is ~5 ms (includes start bit and stop bit).

Parameters: delayTime: The amount of delay (in milliseconds)

Return Value: None

Side Effects: None

void Network_Protocol_EnableTxLineLoading(uint16 loadingTime)

Description: Enables the Network Protocol to turn on the external amplification circuit to load the line for the specified amount of time before it starts the FSK Modulator to transmit the FSK signal.

Parameters: loadingTime: the amount of time (in milliseconds) that the transmit amplification circuit is turned on before the protocol starts transmitting the FSK signal.

Return Value: None

Side Effects: None

void Network_Protocol_DisableTxLineLoading (void)

Description: Disables line loading so that the Network Protocol turns on the amplification circuit at the same time that it starts the FSK Modulator to transmit the FSK signal.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_EnableBiuDetection(void)

Description: Enables the band-in-use detection algorithm, which checks if there is a signal present in the FSK communication frequency (band-in-use) before transmitting. This minimizes the chance of a packet collision from two devices transmitting at the same time. The Network Protocol waits for the line to be free (no band-in-use detection) for a random interval between the range set by the SetBiuWaitInterval API. If the line is free for this interval, the Network Protocol will transmit the packet. If there is a band-in-use detection, the network protocol will select a new random interval and wait for the line to be free for that amount of time. If the total wait time is longer than the Band_In_Use_Timeout_Time parameter, the Network Protocol will stop checking the line and will set the Network_Protocol_BIU_TIMEOUT flag in the return value of the Network_Protocol_ReadTxStatus function.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_DisableBiuDetection(void)

Description: Disables the band-in-use detection algorithm. The Network Protocol will transmit data without checking if the line is free.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_SetBiuThreshold(uint8 thresholdLevel)

Description: Sets the threshold for band-in-use detection. When the amplitude of the received signal is greater than this threshold, it is a band-in-use detection event. The Network Protocol will wait for the amplitude to drop below this threshold for at least the band-in-use wait interval before it transmits the packet.

Parameters: thresholdLevel - The band-in-use detection threshold (in dBuV).

Return Value: None

Side Effects: None

void Network_Protocol_EnableAutoBiuThreshold(void)

Description: Enables the algorithm that sets the BIU threshold to be above the noise floor of the line.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_DisableAutoBiuThreshold(void)

Description: Disables the algorithm that sets the BIU threshold to be above the noise floor of the line.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_SetBiuWaitInterval(uint16 minWaitTime, uint16 maxWaitTime)

Description: Sets the minimum and maximum time (in milliseconds) of the band-in-use detection interval range. Before transmitting a packet, the Network Protocol randomly chooses a value within this range and then checks if the line is free for this entire interval.

Parameters: minWaitTime: the minimum interval (in milliseconds)
maxWaitTime: the maximum interval (in milliseconds)

Return Value: None

Side Effects: The random acquisition time is reset so that it is within the new range

void Network_Protocol_SetBiuTimeout(uint16 maxBIUTimeout)

Description: Sets the total amount of time (in milliseconds) that the Network Protocol can check if the line is free. If this time elapses, then the Network Protocol will set the BIU_TIMEOUT flag in the return value of the Network_Protocol_ReadTxStatus function.

Parameters: maxBIUTimeout: the maximum allowable time (in milliseconds)

Return Value: None

Side Effects: None

void Network_Protocol_EnableFrequencyHopping(uint16 lockTime)

Description: Enables the Network Protocol to automatically switch the FSK Demodulator's center frequency to match the frequency of the received FSK signal. This is determined by comparing the most recent peak reading from the FSK Demodulator versus the noise floor for each frequency. When the peak reading sufficiently exceeds the noise floor, the Network Protocol will stay on that frequency.

Parameters: lockTime: The amount of time (in milliseconds) that the Network Protocol will stay locked on a frequency after it receives a valid packet. It stays locked regardless of the signal level on the line. When non-zero, the assumption is that the transmitter will continue to transmit packets at the same frequency.

Return Value: None

Side Effects: None

void Network_Protocol_DisableFrequencyHopping(void)

Description: Disables frequency hopping. The Network Protocol will not change the FSK center frequency.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_SetMasterFrequency(uint8 fskFrequency)

Description: Sets the active FSK center frequency that is being analyzed. It should match the FSK center frequency of the FSK Demodulator.

Parameters: fskFrequency: the active FSK center frequency.

Options	Description
Network_Protocol_FSK_110K	The FSK center frequency is 110 kHz
Network_Protocol_FSK_132K	The FSK center frequency is 132 kHz

Return Value: None

Side Effects: None

uint8 Network_GetActiveFrequency (void)

Description: Returns the FSK center frequency that is being analyzed. The active frequency is either set by the Network_Protocol_RunFrequencyHop API or the Network_Protocol_SetMasterFrequency API.

Parameters: None

Return Value: The active FSK center frequency.

Options	Description
Network_Protocol_FSK_110K	The FSK center frequency is 110 kHz
Network_Protocol_FSK_132K	The FSK center frequency is 132 kHz

Side Effects: None

uint8 Network_GetNewFskLevel (void)

Description: Returns the latest reading of the peak amplitude (in dB) of the filtered received signal.

Parameters: None

Return Value: The most recent amplitude (in dBuV)

Side Effects: None

uint8 Network_GetMaxFskLevel (void)

Description: Returns the maximum peak amplitude over the interval specified by the Network_Protocol_SetMaxSignalTiming API.

Parameters: None

Return Value: The maximum amplitude (in dBuV)

Side Effects: None

uint8 Network_GetNoiseFloor (void)

Description: Returns the noise floor (in dB) of the line. The noise floor is determined by taking the smallest reading from a set of intervals that each contain the maximum reading over that interval. The assumption is that when there is no data traffic, the noise may vary over time, but it is fairly constant when measured over a longer interval. The scan intervals and the refresh time are set by the Network_Protocol_SetNoiseFloorTiming API.

Parameters: None

Return Value: The noise floor (in dBuV)

Side Effects: None

void Network_SetMaxSignalTiming(uint16 refreshTime)

Description: Sets the amount of time (in milliseconds) that the maximum amplitude is stored for. This is useful when tracking the amplitude of the received FSK signal. When the time is set to be less than the interval between transmitted packets, the result from the Network_Protocol_GetMaxFskLevel will continuously report the same value.

Parameters: refreshTime: the amount of time (in milliseconds) that the maximum amplitude is stored for

Return Value: None

Side Effects: None

void Network_SetNoiseFloorTiming(uint16 scanInterval, uint16 refreshTime)

Description: Sets the timing for measuring the noise floor. When the noise floor is measured, it first calculates the maximum amplitude measured in each scan interval. Then, it finds the minimum of these maximums to determine the noise floor. The scanInterval should be less than half the shortest time between transmitted packets so that at least one interval is guaranteed not to have a data signal. When using band-in-use detection, this time is equal to half the minimum random interval time (Network_Protocol_DEFAULT_BIU_MIN_TIME by default). The refreshTime should be at least as long as the time to transmit the longest packet plus the scanInterval. This ensures that at least one scanInterval with no data on the line will be measured.

Parameters: scanInterval: the time (in milliseconds) that each maximum amplitude is determined
refreshTime: the time (in milliseconds) that the Network Protocol looks to find a minimum from the scan intervals.

Return Value: None

Side Effects: None

void Network_Protocol_InitializeSignalAnalysis(void)

Description: Initializes the receive signal analysis algorithm and data structure.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_ResetSignalAnalysis(void)

Description: Resets the timer that checks if the filter has stabilized so that the proper amplitude is measured.

Parameters: None

Return Value: None

Side Effects: None

uint8 Network_Protocol_RunSignalAnalysis(void)

Description: Collects the statistics (noise floor and maximum) of the filtered received signal. The values are updated over a sliding window defined by the Network_Protocol_DEFAULT_NOISE_SCAN_INTERVAL and UPDATE_MAX_PEAK_TIME parameters. If a new minimum noise floor or maximum peak is found, it will immediately be stored. Otherwise, it will update the minimum with the smallest value over the refresh interval since the last update. The same applies to the maximum.
A new received signal value is available every one millisecond.

Parameters: None

Return Value: The status of updating the statistics.

Options	Description
Network_Protocol_NO_RSSI_UPDATE	The statistics weren't updated
Network_Protocol_RSSI_UPDATED	The statistics were updated

Side Effects: Updates the fskSignalData global structure, which contains information about the noise floor and latest peak values. It is used by the frequency hopping and band-in-use algorithms

void Network_Protocol_InitializeBiuDetection(void)

Description: Initializes the band-in-use detection variables.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_StartAcquirePowerline(void)

Description: Starts the algorithm to check if the line is free. Initializes the state variables.

Parameters: None

Return Value: None

Side Effects: None

uint8 Network_Protocol_GetLineStatus (void)

Description: Checks if the line has been free for the random band-in-use (BIU) interval and returns the status.

Parameters: None

Return Value: The status of updating the statistics.

Options	Description
Network_Protocol_LINE_FREE	The powerline has been free for the interval
Network_Protocol_LINE_WAITING	It is still waiting for the line to be free for the interval
Network_Protocol_LINE_TIMEOUT	The line has been busy for too long. This may be due to noise or too many other devices transmitting on the line

Side Effects: None

void Network_Protocol_PauseAcquirePowerline (void)

Description: Pauses the acquire powerline timers and carrier detection.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_ResumeAcquirePowerline (void)

Description: Restarts the acquire powerline timers and carrier detection.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_StartTxWait(uint16 waitTime)

Description: Starts the timer that tracks the time since the last transmission. This makes sure that if there is another device waiting to transmit, then it will get to transmit before this device.

Parameters: waitTime: The minimum amount of time to wait before transmitting again.

Return Value: None

Side Effects: None

uint8 Network_Protocol_TestFrequencyHop(void)

Description: Tests if the receiver should switch to the other FSK frequency based on the latest FSK amplitude reading and the noise floor.

Parameters: None

Return Value: The status of whether to hop or not.
Network_Protocol_YES_HOP: The receiver should hop to the other frequency
Network_Protocol_NO_HOP: The receiver should stay at the same frequency

Options	Description
Network_Protocol_YES_HOP	The receiver should hop to the other frequency
Network_Protocol_NO_HOP	The receiver should stay at the same frequency

Side Effects: None

void Network_Protocol_StartTimer(void)

Description: Starts the 16-bit system timer. It has a 2 kHz input clock and maximum (65535) period. The Network Protocol automatically starts the timer, but this function is available if the timer is going to be used for some other purpose.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_StopTimer(void)

Description: Stops the 16-bit system timer. The Network Protocol automatically stops the timer when necessary, but this function is available if the timer is going to be used for some other purpose.

Parameters: None

Return Value: None

Side Effects: None

void Network_Protocol_ReadTimer(void)

Description: Returns the current 16-bit value of the system timer. It has a 2 kHz input clock and maximum (65535) period. The Network Protocol automatically reads the timer when necessary, but this function is available if the timer is also going to be used for an additional purpose.

Parameters: None

Return Value: None

Side Effects: None

Sample Firmware Source Code

The code example in the application note AN76458 "PSoC 5LP High Voltage (120 – 240 VAC) Powerline Communication" contains the Network Protocol, along with the FSK Modulator and FSK Demodulator components. This forms a complete powerline communication (PLC) system and demonstrates how to read an analog voltage with an ADC, transmit that value over the powerline, then receive the value and display it on an LCD.

Functional Description

The purpose of the Network Protocol is to ensure that the correct data is sent to the intended destination, in a timely manner. Since a powerline is a bus, it is also important to prevent data collisions. Figure 3 shows the schematics of the Network Protocol component. It uses a UART, 16-bit Timer, and a Control Register. It runs a state machine to co-ordinate the transmission and reception of data.

UART

When the application wants to transmit data, the Network Protocol encapsulates the data in a packet. The UART component converts each byte of the packet into a serial bit stream for modulation by the FSK Modulator. In the receive direction, the UART converts the serial bit stream from the FSK Demodulator into the bytes of the packet. The source clock must be eight times the desired baud rate. If the UART_Clock_Source parameter is Internal, the Network

Protocol sets the UART_Clock's frequency to be eight times the value of the Baud_Rate parameter. If the UART_Clock_Source parameter is External, the user must set the uclk clock input's frequency to be eight times the value of the Baud_Rate parameter.

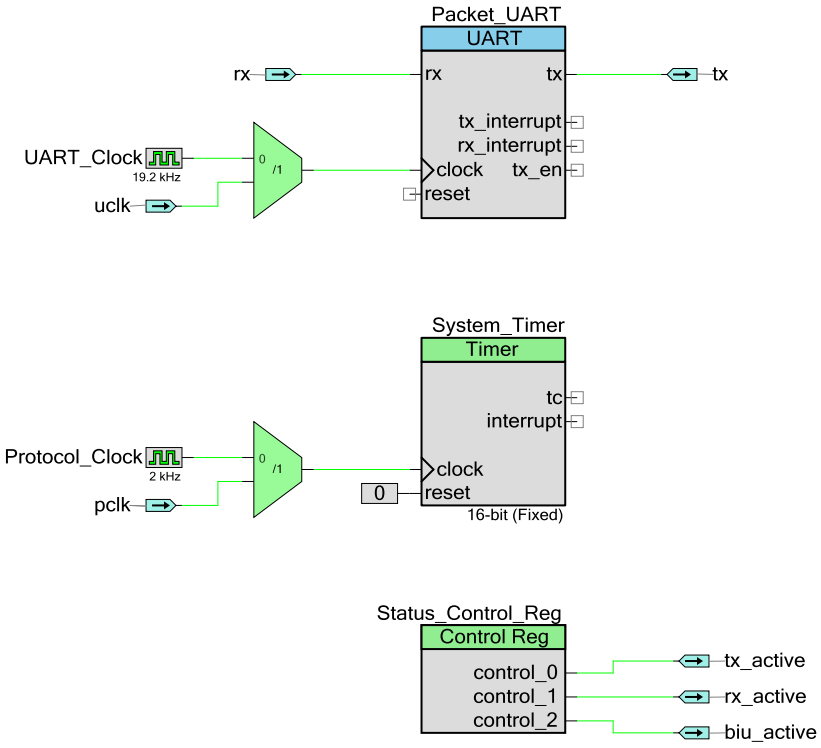
Timer

The 16-bit Timer component tracks the timing for band-in-use detection, transmit startup, receive timeouts, acknowledgment timeouts, and receive signal analysis. If the Protocol_Clock_Source is External, the user must set the Protocol_Clock_External_Frequency parameter to match the nearest integer (in kHz) of the pclk clock input's frequency.

Control Register

The Control_Reg component drives each digital output according to the state of the protocol. When the protocol is transmitting data, the tx_active output is high. When the protocol receives a byte of data that matches the Cypress packet structure, the rx_active output is high. Up until the destination address is received, this output pulses high for less than 20 us after each received byte. If the destination address of the received packet matches the local address, this output stays on until the device receives the last byte or an invalid byte. When the network protocol is checking if the line is busy, the biu_active output is high when there is a band-in-use detection (see Band-In-Use Detection for details).

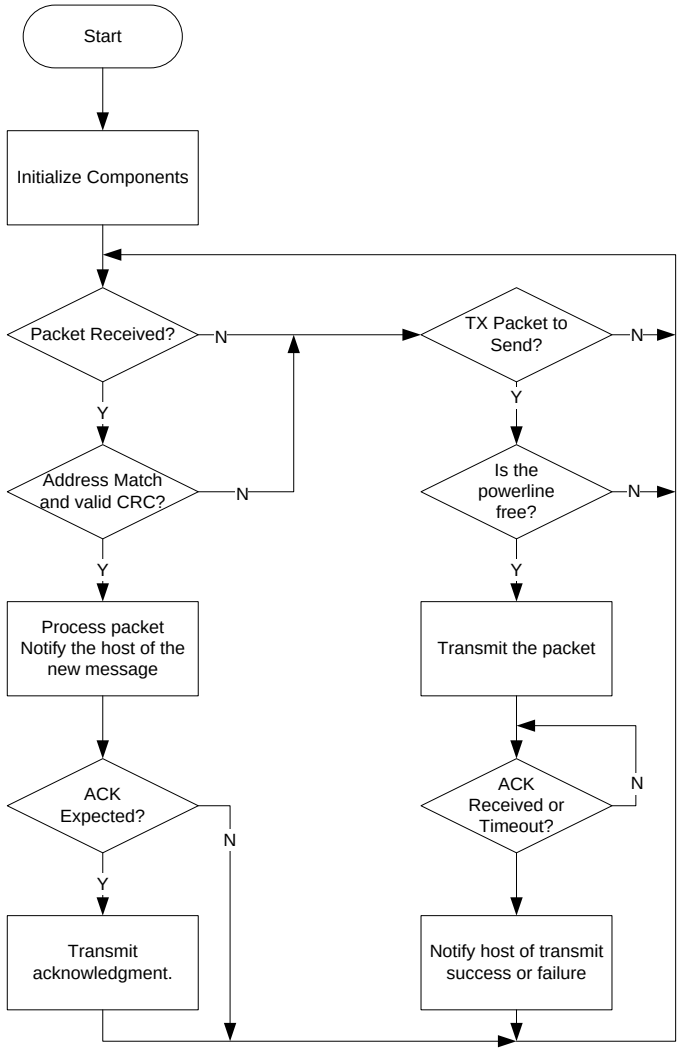
Figure 3. Network Protocol Schematics



Network Protocol State Machine

The flow chart of the network protocol state machine is shown in Figure 4. Since this is a half-duplex system, the device is either idle, receiving a packet, transmitting a packet, or checking if the line is free. If the device receives a packet, the protocol evaluates whether the destination address of the packet matches its own address. It also determines whether the calculated CRC matches the CRC value in the packet. When both are valid, the protocol notifies the application layer that a new message is available. If the packet indicates that an acknowledgment is expected, then the protocol transmits an acknowledgment packet back to the source of the message.

Figure 4. Network Protocol Flow Diagram



If the application has a message to send, the protocol first checks if the powerline is free. This is known as band-in-use (BIU) detection. If the line is free, the protocol transmits the packet and

then waits for an acknowledgment from the destination. If the device receives an acknowledgment or too much time elapses, the protocol notifies the application of the status of transmission.

The network protocol is backwards compatible with the CY8CPLC20 PSoC 1 device.

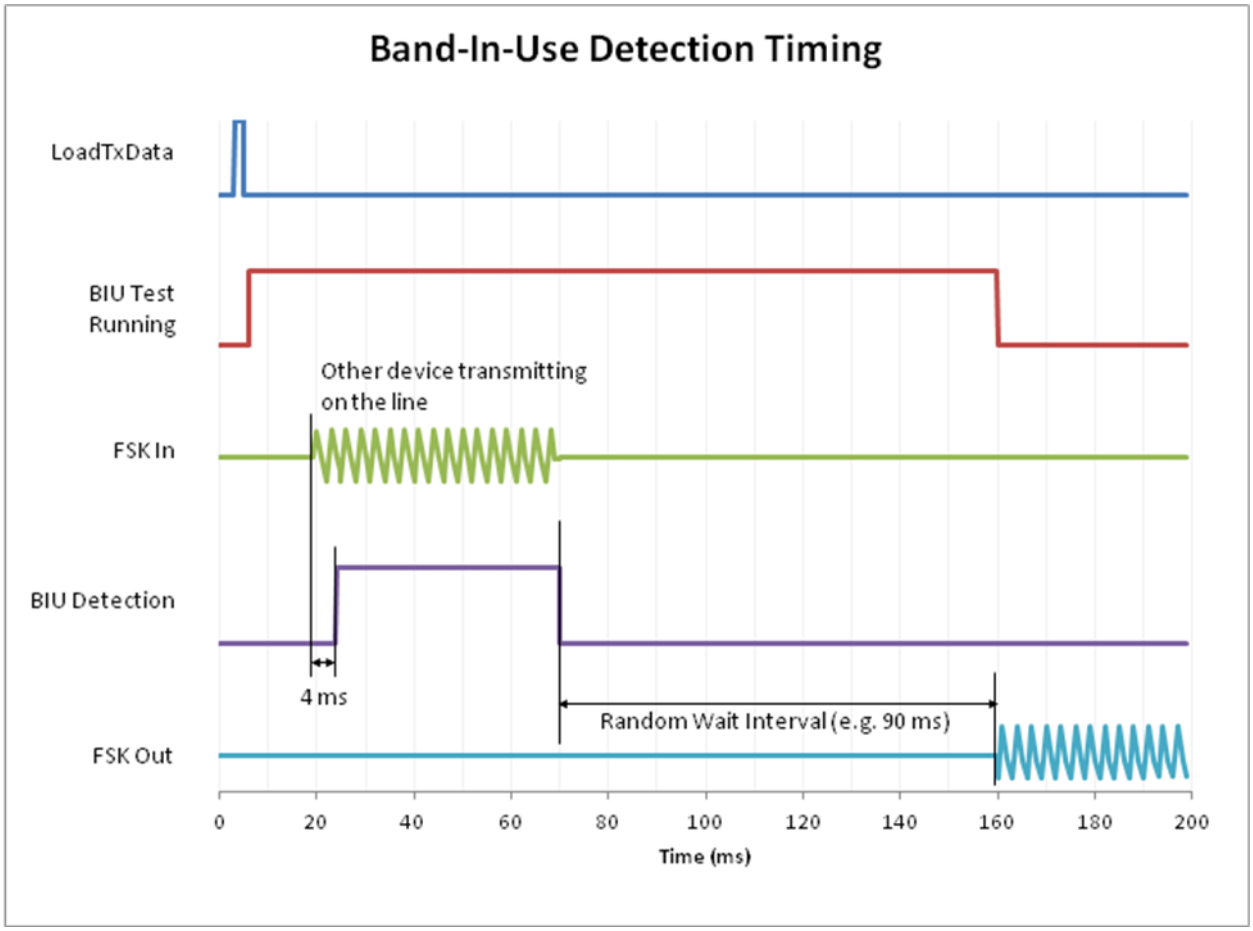
Band-In-Use Detection

The CENELEC EN50065-2-1 specification sets timing and threshold levels to detect a band-in-use (BIU) condition. The Cypress protocol meets this specification and has the option to change the detection levels for non-CENELEC compliant systems.

For CENELEC compliant systems, the protocol selects a random period between 85 and 115 ms (out of 7 possible values in this range). The line must be free for this entire period to transmit the message. If a BIU detection occurs, the timer restarts. A BIU detection event, as defined under CENELEC EN 50065-1, occurs whenever a signal in the range of 131.5 kHz to 133.5 kHz exceeds 86 dBµVrms for at least 4 ms. Also, the last device to transmit must wait for at least 125 ms to transmit again. This ensures that any other device that is waiting to transmit will get the next turn. Figure 4 provides an example of the band-in-use timing. The LoadTxData signal represents the time when the application calls the Network_Protocol_LoadTxData API followed by the Network_Protocol_Poll API. The protocol will immediately start testing if the line is free. When there is a signal detection on the input (FSK In), the protocol first waits for 4 ms to make sure that it is a valid detection. Once there is a valid BIU detection, the protocol waits for the signal level to drop below the threshold. When the input signal level drops, the protocol will wait for the random wait interval. If that interval elapses and there is no band-in-use detection, then the protocol will start transmitting the packet.

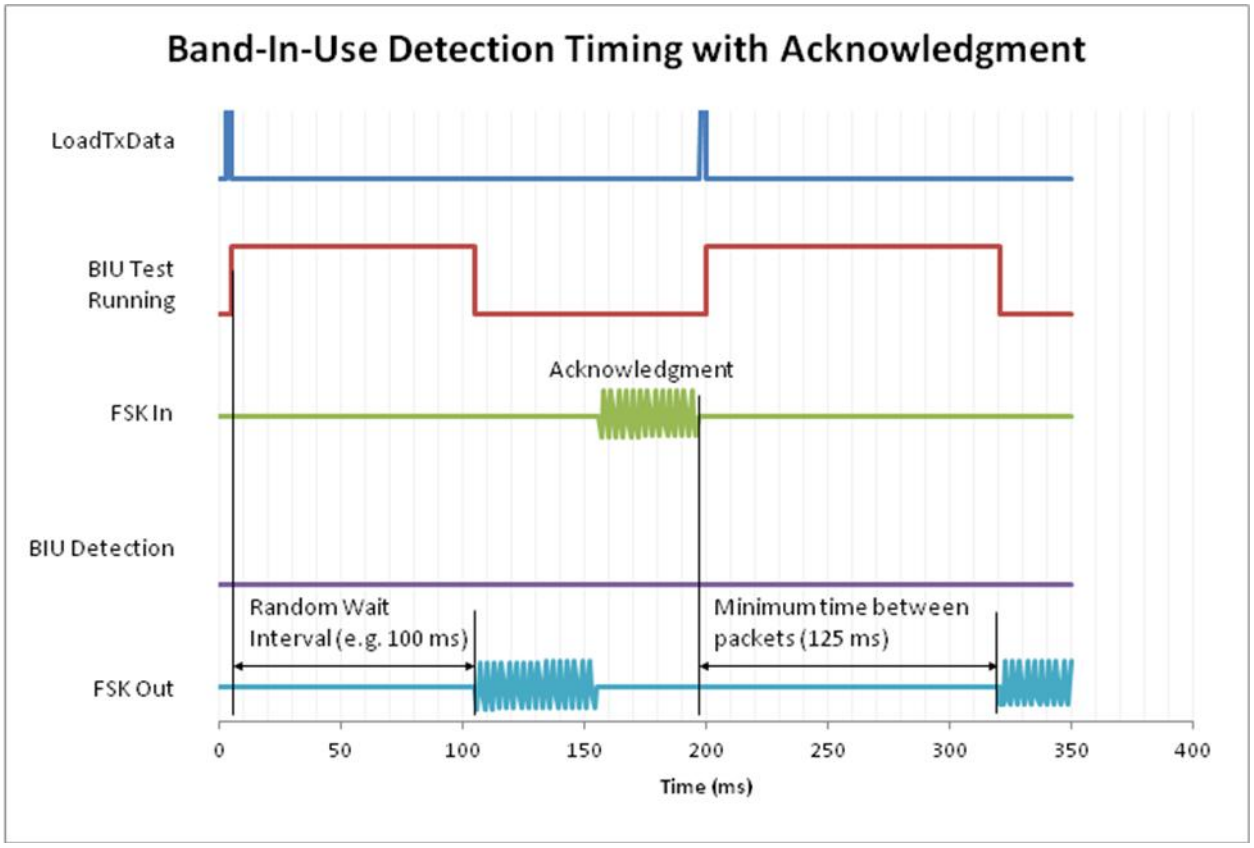
If the application calls the Network_Protocol_ReadTxStatus() function while the protocol is testing if the line is free, the function will return with the flag Network_Protocol_BIU_IN_PROGRESS. When the line is free, it will return with the flag Network_Protocol_TX_IN_PROGRESS.

Figure 5. Band-In-Use Detection Timing Diagram



When there is a demand-response interaction between two devices (for example, one device sends a packet and the other sends back an acknowledgment), the responding device does not check if the line is free before transmitting the response. After the initial packet is transmitted, all other devices on the line must wait for at least 85 ms before transmitting. Therefore, if the responding device immediately starts transmitting back an acknowledgment, there is no risk of a packet collision. This assumes that only one device will acknowledge the packet (each device on the line has a unique address). The minimum wait time of 125 ms applies to a group of transmitters that have a demand-response interaction. Therefore, the wait time starts after the originating device receives the acknowledgment. Figure 6 provides an example of the timing when acknowledgments are transmitted.

Figure 6. Band-In-Use Detection with Acknowledgment Timing Diagram



For non-CENELEC compliant systems, the application can modify the BIU detection period and threshold. This is useful for systems that require lower latency or a fixed latency to transmit messages. The Network_Protocol_SetBiuWaitInterval() function sets the minimum and maximum possible interval period.

The Network_Protocol_SetBiuThreshold() function sets the detection threshold. The Network_Protocol_EnableAutoBiuThreshold() function will automatically set the detection threshold to be 6 dB above the existing noise floor on the line.

By default, if 1.1 seconds elapse and the line is not free, the protocol stops line acquisition and notifies the application by setting the Network_Protocol_BIU_TIMEOUT flag when the application calls the Network_Protocol_ReadTxStatus() function. The application can change the BIU timeout time by calling the Network_Protocol_SetBiuTimeout() function.

Band-In-Use Detector Implementation

The protocol calls the FSK_Demodulator_ReadFskPeak() function to get the latest amplitude of the input signal from the FSK Demodulator. The protocol uses a 16-bit timer to make sure that the FSK Demodulator has collected enough samples before the protocol will use the result from the FSK Demodulator. Once it has a valid amplitude, it calls the

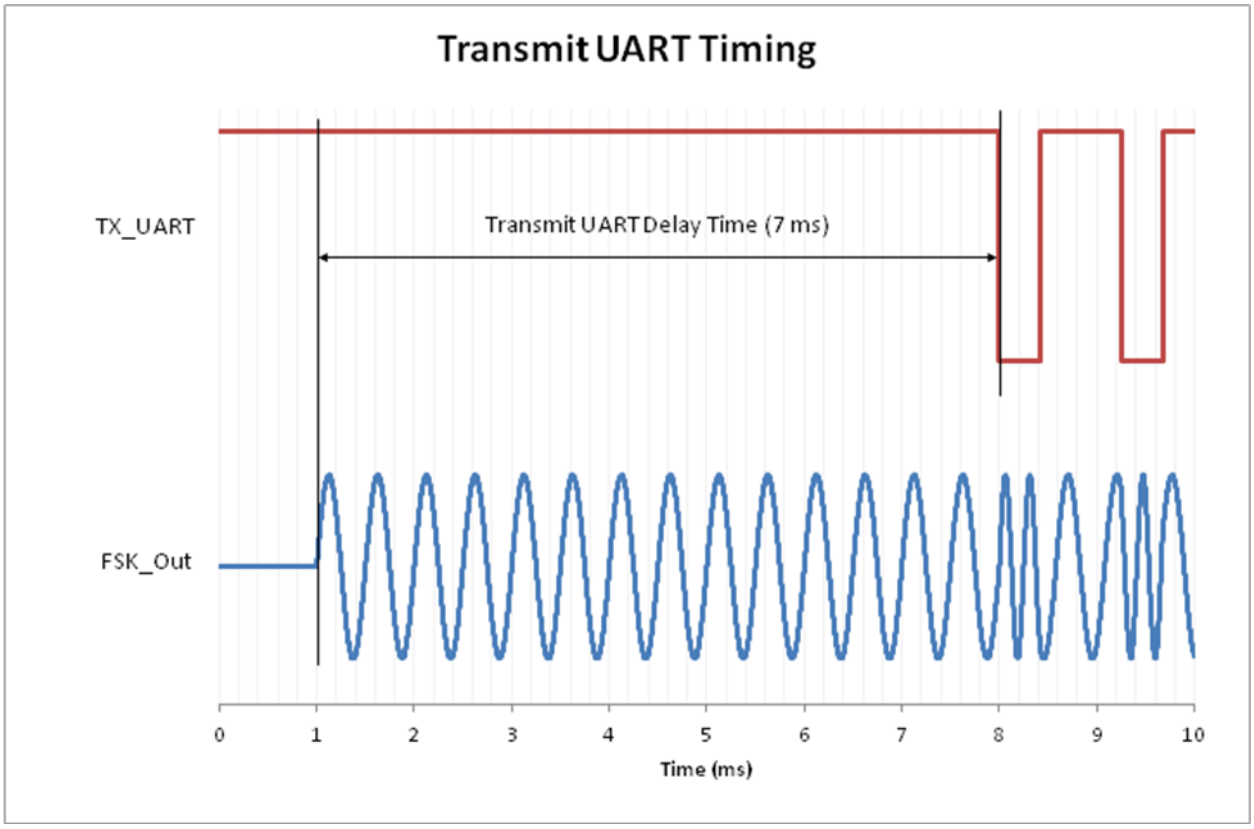
FSK_Demodulator_ResetFskPeak() function so that the FSK Demodulator can measure a new amplitude. The timer also tracks the band-in-use interval and the timeout.

Packet Transmission

When the application calls the Network_Protocol_LoadTxData() function, the protocol forms a PLC packet (see the Cypress PLC Packet Structure) from the user data and the address information. The Network_Protocol_SetLogicalAddress(), Network_Protocol_SetSourceAddressType(), and Network_Protocol_SetDestinationAddress() functions set the address information.

When the line is free, the protocol will start the FSK Modulator and then call the UART to transmit the first byte. There is a delay from when the FSK Modulator starts and when the UART starts transmitting. During this time, the FSK Modulator transmits the FSK signal for a logic ‘1’. Since the receiver is always demodulating the input signal, the receiver’s UART might be in the middle of receiving a byte from noise. The FSK Modulator’s signal will be greater than the noise, so the UART will start receiving a logic ‘1’ signal. This will give it time to finish up receiving the false data and be prepared for the first start bit of the packet. The minimum amount of the delay is the time from the start of one byte to the start of the next. When the UART transmits at 2400 bps with no parity, it takes 5 ms from the start of one byte to the next. By default this UART delay time is 7 ms. The Network_Protocol_SetUartDelayTime() function changes the delay time. Figure 7 shows the timing between the FSK Modulator and the UART at the start of transmission.

Figure 7. Transmit UART Timing Diagram



When the application calls Network_Protocol_Poll, the protocol will load the bytes of the packet into the UART whenever the UART is not full (four byte buffer). When the UART finishes the transmission, the next protocol state is based on whether the protocol expects an acknowledgment. If it doesn't expect an acknowledgment, then it notifies the host by setting the Network_Protocol_TX_COMPLETE flag when the application calls the Network_Protocol_ReadTxStatus() function. If it expects an acknowledgment, then it waits for receipt of the acknowledgment. If it receives the acknowledgement, it notifies the host by setting the Network_Protocol_TX_COMPLETE flag when the application calls the Network_Protocol_ReadTxStatus() function. If there is no acknowledgment within the acknowledgment timeout window, the protocol notifies the host by setting the Network_Protocol_ACK_TIMEOUT flag when the application calls the Network_Protocol_ReadTxStatus() function.

The Network_Protocol_EnableAck() function makes the protocol expect acknowledgments and sets the acknowledgment timeout window. The Network_Protocol_DisableTxAck() function makes the protocol not expect acknowledgments.

Packet Reception

When the UART receives a byte, it stores it in its four-byte deep buffer. When the application calls Network_Protocol_Poll, it runs a state machine that waits for the preamble (0xAB) and once received, it stores the subsequent bytes in the receive buffer. It continues to run until one of the following conditions occurs:

- The entire packet is received (the length is defined in the PAYLOAD LENGTH byte of the packet).
- (Logical or physical destination address type) The destination address does not match this protocol's local address.
- (Group destination address type) The group address does not match a group that the protocol is a member of.
- There is an error in the packet.
- A byte hasn't been received within the timeout window (8 ms by default); or
- There is a packet in the receive buffer that the application hasn't read. In this case, the protocol notifies the application by setting the Network_Protocol_RX_OVERFLOW flag when the application calls the Network_Protocol_ReadRxStatus() function.

If an acknowledgment is expected (ACK MODE = '1' in the packet header), the protocol transmits an acknowledgment back to the source.

If the packet is a duplicate (same sequence number and CRC), the protocol does not process it and it does not forward it to the application. The protocol still sends the acknowledgment to inform the source that it received the packet.

If the packet is new (i.e. not a duplicate), the protocol notifies the application by setting the Network_Protocol_RX_COMPLETE flag when the application calls the Network_Protocol_ReadRxStatus() function. In addition, the protocol stores the command ID, data payload, the payload length, the addressing type, and the source address. The application can retrieve this information by calling the Network_Protocol_GetRxCommandID (), Network_Protocol_GetRxData(), Network_Protocol_GetRxDataLength(), Network_Protocol_GetRxDestinationAddressType(), Network_Protocol_GetRxSourceAddressType(), and Network_Protocol_GetRxSourceAddress() functions.

If the packet's command ID is a PLC control message (command ID < 0x10), the protocol will process it, respond (if necessary), and notify the application.

When the application calls the Network_Protocol_ReadRxStatus() function, it automatically clears the receive status flags. This allows the protocol to receive a new packet.

Cypress PLC Packet Structure

Table 1 shows the format of a Cypress PLC packet. The protocol transmits this packet over the powerline. The general structure of the packet contains a variable length header and variable length data (payload). The header size can range from a minimum of 7 bytes to a maximum of 21 bytes, depending on the addressing type. The payload length can vary from a minimum of 0 bytes to a maximum of 31 bytes.

Table 1. Structure of a PLC Packet

Bit Offset							
7	6	5	4	3	2	1	0
PREAMBLE (0xAB)							
SA TYPE	DA TYPE		ACK MODE	RSVD	RSVD	RESPONSE	RSVD
DESTINATION ADDRESS (up to 8 bytes)							
SOURCE ADDRESS (up to 8 bytes)							
COMMAND ID							
RSVD				PAYLOAD LENGTH			
SEQ NUM				HEADER CRC			
PAYLOAD (0 up to 31 Bytes)							
PACKET CRC							

Packet Header

The header comprises at least the first seven bytes of the packet. It contains the fields described in the following sections. Unused fields that are marked as RSVD are transmitted as 0s.

Preamble

A one-byte preamble (0xAB) identifies the start of the packet. The receiver must first receive this byte before it starts to store the packet in its buffer.

Control Byte

The Control Byte field is one byte. It contains bit fields of the addressing type, acknowledgment mode and response type of the packet.

SA TYPE

The SA TYPE field is one bit (bit 7). It indicates the type of addressing in the Source Address field (Table 2). For the logical addressing type, the Network_Protocol_SetLogicalAddressMode function sets whether it is 8-bit or 16-bit. A device can only support one or the other logical addressing mode, but not both at the same time.

Table 2. Source Address Type

SA TYPE	Description
0	Logical (8-bit or 16-bit)
1	Physical (64-bit)

DA TYPE

The DA TYPE field is two bits (bits 5 and 6). It represents the type of addressing in the Destination Address field (Table 3).

Table 3. Destination Address Type

DA TYPE	Description
00	Single node logical
01	Group logical
10	Single node physical
11	Reserved

ACK MODE

The ACK MODE field is one bit (bit 4). It defines whether the source device expects to receive an acknowledgment from the destination (Table 4). Typically, a device enables acknowledgment mode when it is critical that the source knows that the destination receives the message. The drawback of acknowledgment mode is that the overall throughput slows down because an acknowledgment packet occupies the powerline. An acknowledgment packet uses the same structure as a normal PLC packet (Table 1) with the RESPONSE bit set to ‘1’ and the packet length set to ‘0’. The source interprets the failure to receive an acknowledgement within the selected timeout window (set by the Network_Protocol_EnableAck function) as a high probability that a packet failed in the transmission. The protocol supports an option to automatically re-transmit the packet (set by the Network_Protocol_SetTxRetries function).

Table 4. Acknowledgment Mode

ACK MODE	Description
0	Acknowledgment Not Expected
1	Acknowledgment Expected

RESPONSE

The RESPONSE field is one bit (bit 1). It indicates that the packet is a response to a previous command. The COMMAND ID in the response packet matches the same COMMAND ID as the initial packet.

If the command sent requires a response with data, the data is in the PAYLOAD portion of the response packet. This response packet serves as the acknowledgement. If the command sent does not require response data and ACK MODE is set in the transmitted packet, the acknowledgement is sent back as a zero-length data packet, containing the original command code, with the RESPONSE bit set.

DESTINATION ADDRESS

The DESTINATION ADDRESS field is from one to eight bytes. The length depends on the DA TYPE and logical address mode. It contains the logical, group, or physical address, of the intended node.

SOURCE ADDRESS

The SOURCE ADDRESS field is from one to eight bytes. The length depends on the SA TYPE and logical address mode. It contains the logical or physical address of the node that transmits the packet.

COMMAND ID

The COMMAND ID field is one byte. It contains the 8-bit command to be sent to the destination. Command IDs 0x01 to 0x0F are commands that the network protocol automatically processes. Command IDs outside of this range are user-defined commands that the network protocol passes directly to the application.

Table 5. Command ID List

ID	Name	Description	Payload (TX Data)	Response (RX Data)
0x01	CMD_SET_TX_ENABLE	Enable or disable the transmitter	0 - Disables the transmitter 1 - Enables the	If configuration access is enabled, Response = 00 (Success)

Table 5. Command ID List

ID	Name	Description	Payload (TX Data)	Response (RX Data)
			transmitter	If configuration access is disabled, Response = 01 (Denied)
0x03	CMD_SET_EXT_MODE	Sets the logical addressing mode	0 - Disable Extended Addressing 1 - Enable Extended Addressing	If configuration access is enabled, Response = 00 (Success)
				If configuration access is disabled, Response = 01 (Denied)
0x04	CMD_SET_LOGICAL_ADDRESS	Set the logical address to the value in payload	If the logical addressing mode is 8-bit, Payload = 8-bit Logical Address If the logical addressing mode is 16-bit, Payload = 16-bit Logical Address	If configuration access is enabled, Response = 00 (Success)
				If configuration access is disabled, Response = 01 (Denied)
0x05	CMD_GET_LOGICAL_ADDRESS	Request the logical address	None	If the logical addressing mode is 8-bit, Response = 8-bit Logical Address
				If the logical addressing mode is 16-bit, Response = 16-bit Logical Address
0x06	CMD_GET_PHYSICAL_ADDRESS	Request the physical address	None	64-bit physical address
0x07	CMD_GET_PLC_MODE	(Legacy support for PSoC 1) Gets the PLC_MODE register value	None	PLC_MODE value
0x08	CMD_GET_FW_VERSION	Request the firmware version	None	8-bit firmware version
0x09	CMD_DATA	Transmit the data in the payload	Payload = Local transmit Data	If ACK MODE = 0, Response = None
				If ACK MODE = 1, Response = Acknowledgment
0x0A	CMD_REQUEST_DATA	Send the payload along with a request for a response. Wait for the response for up to 1.5 seconds	Payload = Local transmit Data	Response = CMD_RESPONSE command .
0x0B	CMD_RESPONSE_DATA	Send the payload as a response to the CMD_REQUEST_DATA command	Payload = Local transmit Data	None
0x0C	CMD_SET_BIU_DISABLE	Enables/Disables Band-In-Use (BIU) functionality at the remote node	0 - Enable Remote BIU 1 - Disable Remote BIU	If configuration access is enabled, Response = 00 (Success)
				If configuration access is disabled, Response = 01 (Denied)

Table 5. Command ID List

ID	Name	Description	Payload (TX Data)	Response (RX Data)
0x0D	CMD_SET_BIU_THRESHOLD	Sets the BIU Threshold value at the remote node	0 – 7 – Set the threshold to 45 + (payload * 6) 8 – 15 - Set the threshold to 81 + ((payload – 6) * e) >15 – Set the threshold to the payload Note: 0-15 is for legacy support of PSoC 1	If configuration access is enabled, Response = 00 (Success) If configuration access is disabled, Response = 01 (Denied)
0x0E	CMD_SET_GROUPS	Sets the Group Membership of the remote node	Byte0 - Remote 8-bit Single Group Membership Address Byte1- Remote Multiple Group Membership Address	If configuration access is enabled, Response = 00 (Success) If configuration access is disabled, Response = 01 (Denied)
0x0F	CMD_GET_GROUPS	Gets the Group Membership of the remote node	None	Byte0 - Remote Single Group Membership Address Byte1- Remote Multiple Group Membership Address
0x10 to 0x2F	Reserved			
0x30 to 0xFF	User Defined Command Set		Payload = Local TX Data	If ACK MODE = 0, Response = None If ACK MODE = 1, Response = Acknowledgment

LENGTH

The LENGTH field is 5 bits. It contains the number of bytes of data in the PAYLOAD field.

SEQ NUM

The SEQ NUM field is 4 bits. It provides a unique identifier for each packet sent. The protocol increments this number for each new packet sent. The number stays the same when the protocol re-transmits the same packet. If the destination receives consecutive packets with the same sequence number and CRC, it assumes that it is a duplicate packet and it does not notify the host. It still transmits an acknowledgment so that the source knows that the message was received.

HEADER CRC

The HEADER CRC field is 4 bits. It contains a Cyclic Redundancy Check (CRC) value that the protocol calculates from the complete header (not including the preamble). The CRC algorithm uses the CRC-4-ITU polynomial ($x^4 + x + 1$). Its purpose is to allow the receiver to suspend receiving the rest of the packet when the header is corrupted.

PAYLOAD

The PAYLOAD field is 0 to 31 bytes. The LENGTH field in the packet header defines the length of the payload. It contains the message.

PACKET CRC

The PACKET CRC field is one byte. It is the last byte of the package and contains the 8-Bit CRC value used to check the data integrity of the packet. The CRC algorithm uses the CRC-8-CCITT polynomial ($x^8 + x^7 + x^3 + x^2 + 1$). The CRC calculation includes the header (except for the preamble) and payload portions of the packet.

Forward Error Correction

Forward error correction (FEC) can help reduce the number of packet errors by encoding each byte of data in a way that enables the protocol to detect and correct a single bit error in the encoded byte. The protocol cannot correct more than one bit error within a byte and instead uses the CRC to detect multi-bit errors.

If the application calls the Network_Protocol_EnableFec() function and transmits a packet, the protocol encodes the entire packet (including the preamble and CRCs). The FEC encoding table takes each 4-bit nibble of the byte and encodes it into a 7-bit value. The eighth bit is set to ‘0’ for all cases, except for the first element where it is set to ‘1’. The encoding table is in the FECtableCoder array, as shown in Code 1.

Code 1. FEC Encoding Table

```
    const uint8 FECtableCoder[16] =
    {
                0x80, 0x07, 0x19, 0x1E, 0x2A, 0x2D, 0x33, 0x34,
                0x4B, 0x4C, 0x52, 0x55, 0x61, 0x66, 0x78, 0x7F
};
```

The protocol decodes each received byte with the array FECtableDecoder as a look-up table, as shown in Code 2. The protocol uses only the lower seven bits of the received byte to index the table. The decoded value is a 4-bit nibble in the lower half of the returned byte.

Code 2. FEC Decoding Table

```
    const uint8 FECtableDecoder[128] =
    {
                0x00, 0x00, 0x00, 0x01, 0x00, 0x01, 0x01, 0x01,
                0x00, 0x02, 0x04, 0x08, 0x09, 0x05, 0x03, 0x01,
                0x00, 0x02, 0x0A, 0x06, 0x07, 0x0B, 0x03, 0x01,
                0x02, 0x02, 0x03, 0x02, 0x03, 0x02, 0x03, 0x03,
                0x00, 0x0C, 0x04, 0x06, 0x07, 0x05, 0x0D, 0x01,
                0x04, 0x05, 0x04, 0x04, 0x05, 0x05, 0x04, 0x05,
                0x07, 0x06, 0x06, 0x06, 0x07, 0x07, 0x07, 0x06,
                0x0E, 0x02, 0x04, 0x06, 0x07, 0x05, 0x03, 0x0F,
                0x00, 0x0C, 0x0A, 0x08, 0x09, 0x0B, 0x0D, 0x01,
                0x09, 0x08, 0x08, 0x08, 0x09, 0x09, 0x09, 0x08,
                0x0A, 0x0B, 0x0A, 0x0A, 0x0B, 0x0B, 0x0A, 0x0B,
                0x0E, 0x02, 0x0A, 0x08, 0x09, 0x0B, 0x03, 0x0F,
                0x0C, 0x0C, 0x0D, 0x0C, 0x0D, 0x0C, 0x0D, 0x0D,
                0x0E, 0x0C, 0x04, 0x08, 0x09, 0x05, 0x0D, 0x0F,
                0x0E, 0x0C, 0x0A, 0x06, 0x07, 0x0B, 0x0D, 0x0F,
                0x0E, 0x0E, 0x0E, 0x0F, 0x0E, 0x0F, 0x0F, 0x0F
    };
```

If the received packet expects an acknowledgment, the protocol will encode the acknowledgement only if it received an FEC encoded packet. Therefore, the protocol ignores its local FEC setting when it transmits acknowledgments.

Receive Signal Analysis

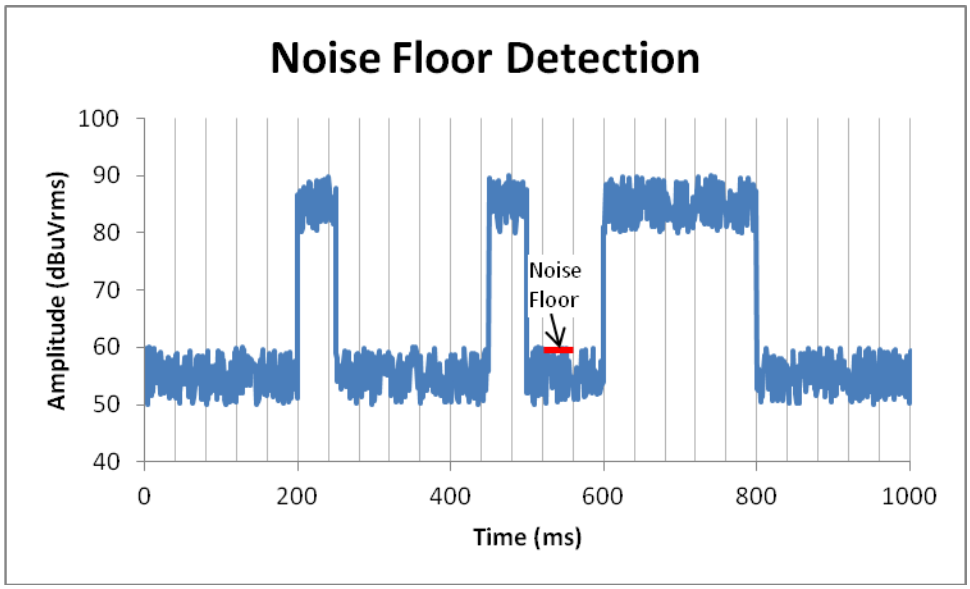
The Network Protocol has the ability to analyze the receive signal that is measured by the FSK Demodulator. The most important values that it determines are the latest peak, the maximum peak and the noise floor. The Network_Protocol_GetNewFskLevel (), Network_Protocol_GetMaxFskLevel (), and Network_Protocol_GetNoiseFloor () return each of these values, respectively. These values are useful for automatically setting the BIU threshold, for frequency hopping, and for debugging.

The recent maximum peak is the largest value recorded over a sliding window. If it reads a new sample that is greater than the recent maximum, it immediately stores it and re-starts the window. If it doesn't find a value that is higher than the current maximum, it will update the maximum with the largest value that it measured during that window. The Network_Protocol_SetMaxSignalTiming() function sets the duration of the sliding window. The duration should be greater than the amount of time between packets when the application is constantly sending data. That way, the maximum value will consistently report the amplitude of the signal on the line. The longest amount of time between packets occurs when an acknowledgment isn't received and the random BIU interval is at the longest duration. In this

case, the acknowledgment timeout takes ~120ms, the maximum BIU interval is ~115ms, which totals 235ms.

The challenge with finding the noise floor is being able to differentiate between PLC signals and noise, since the PLC signal can be very close to the noise floor. To do this, the algorithm uses the rules defined by CENELEC, which define when a device can transmit a PLC signal. Assuming that the system is using the default band-in-use configuration (which is CENELEC compliant), each device must wait at least 85 ms between the end of one sequence of transmissions (transmit and ACK) and the start of the next sequence. In addition, a device cannot transmit continuously for more than 1.1 s. Therefore, if the protocol measures the maximum amplitude of the line during that 85 ms gap, it will be able to determine the noise floor over any span of 1.1 s. Since the protocol doesn't necessarily know when another PLC transmission ends, the algorithm records the maximum value over a 40 ms interval, which is slightly less than half of the minimum gap. That way, even if it measured the tail end of a PLC transmission during one interval, it would measure only noise during the next interval. It then looks over a 1.1 s span and takes the minimum of these 40ms maximums to determine the noise floor. Figure 8 shows how the noise floor is the minimum of each of the maximums that are sampled in 40ms intervals (grey lines). The Network_Protocol_SetNoiseFloorTiming () function sets the scan interval and the refresh span. This is useful if using non-CENELEC values for the time between packets.

Figure 8. Signal Amplitude Noise Floor



Frequency Hopping

The purpose of frequency hopping is for two devices to automatically switch to the same carrier frequency and successfully transmit and receive data over the powerline. The transmitter and receiver have different methodologies for selecting the frequency to communicate at.

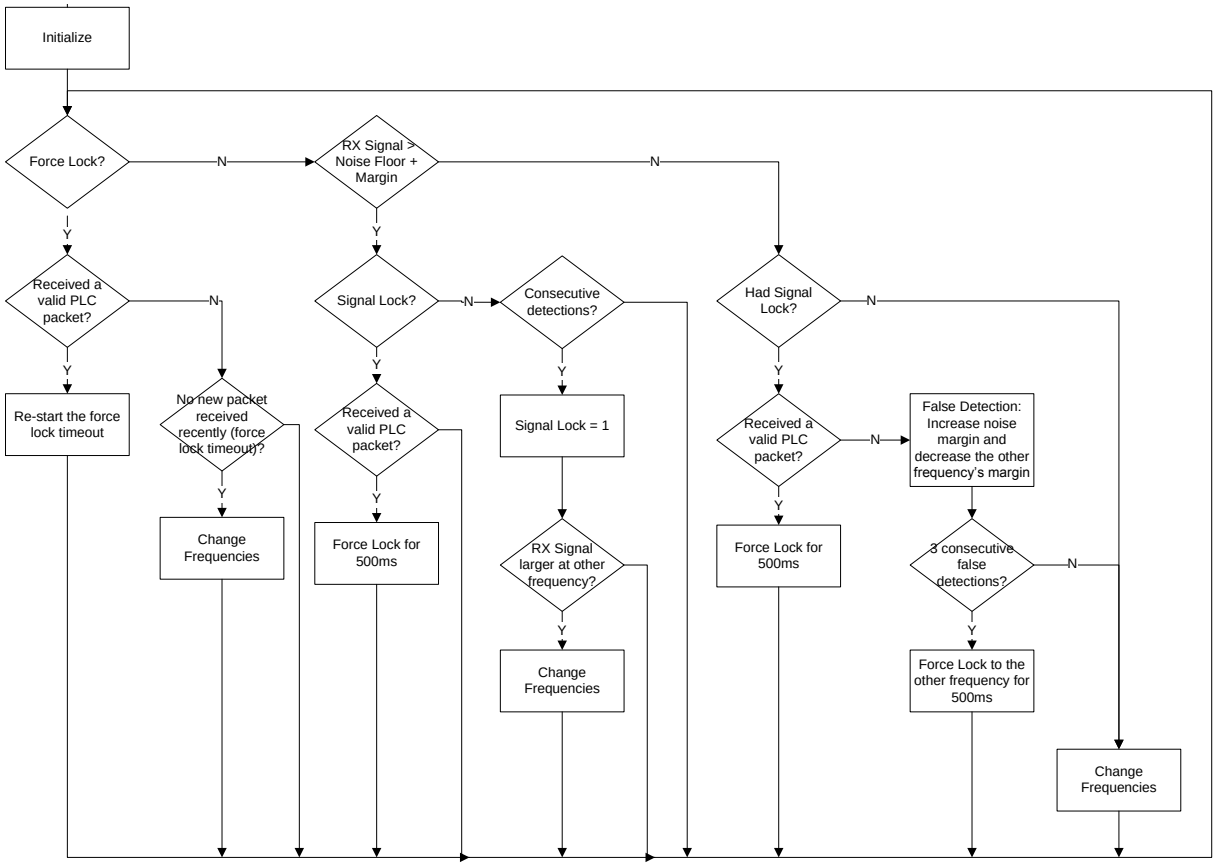
At a high-level, the receiver hops between the available FSK center frequencies until it sees a signal that exceeds the noise floor. It then locks on to this frequency until the signal drops below the noise floor. If it receives a PLC packet, it will stay locked on that frequency until a timeout occurs. The transmitter hops to find the frequency that gives the best success rate.

The Network Protocol component performs frequency hopping for only the receiver because the factors that determine the transmitter's frequency selection are application dependent (success rates). The application note AN76458 "PSoC 5LP High Voltage (120 – 240 VAC) Powerline Communication" provides an example of transmitter frequency hopping.

A more detailed flow of the receiver frequency hopping is shown below. It hops between the 110 kHz and 132 kHz FSK center frequencies. The key features are:

1. If the received signal exceeds the noise floor (plus a margin) for two consecutive readings, then it will check if the signal is larger at the other frequency. It will then temporarily lock to whichever frequency has a higher reading. The check at the other frequency is needed because the receive filter doesn't completely filter out the other frequencies, so for example, a 120dB PLC signal @ 132kHz may look like an 80dB signal @ 110 kHz. The receiver may have been at 110 kHz when the PLC signal was received and it would see the 80 dB signal as a detection. When it switches over to 132kHz, it will see a larger signal and lock to 132 kHz.
2. After it locks, if the signal falls below the noise floor and a valid packet header wasn't received, then it is treated as a false detection. To minimize false detections, the device increases the margin that is added to the noise floor when detecting a signal. At the same time, it decreases the margin of the other frequency so that the noisier channel will have the higher noise margin. Even after the higher noise margin, if there are multiple false detections, then it will force a lock to the other frequency.
3. If the receiver receives a packet with a valid packet header, then it will force the receiver to lock to that frequency, regardless of the signal level on the line. It will stay locked on that frequency for the lock time, which is the parameter in the Network_Protocol_EnableFrequencyHopping() function . The protocol restarts the timeout when it receives a new packet. It also will force a lock when the other frequency has too many false signal detections. The lock time should be longer than the duration between two packets so that even if it drops one packet, it will stay locked to receive the second packet. For example, if the packet time is ~70ms, the acknowledgment timeout is ~50ms and the maximum BIU interval is ~115ms. The duration between two packets is the sum of two acknowledgment timeouts, two BIU intervals, and one packet time, which is 400ms.
4. If it has not locked to a frequency, then it will switch to the other frequency.

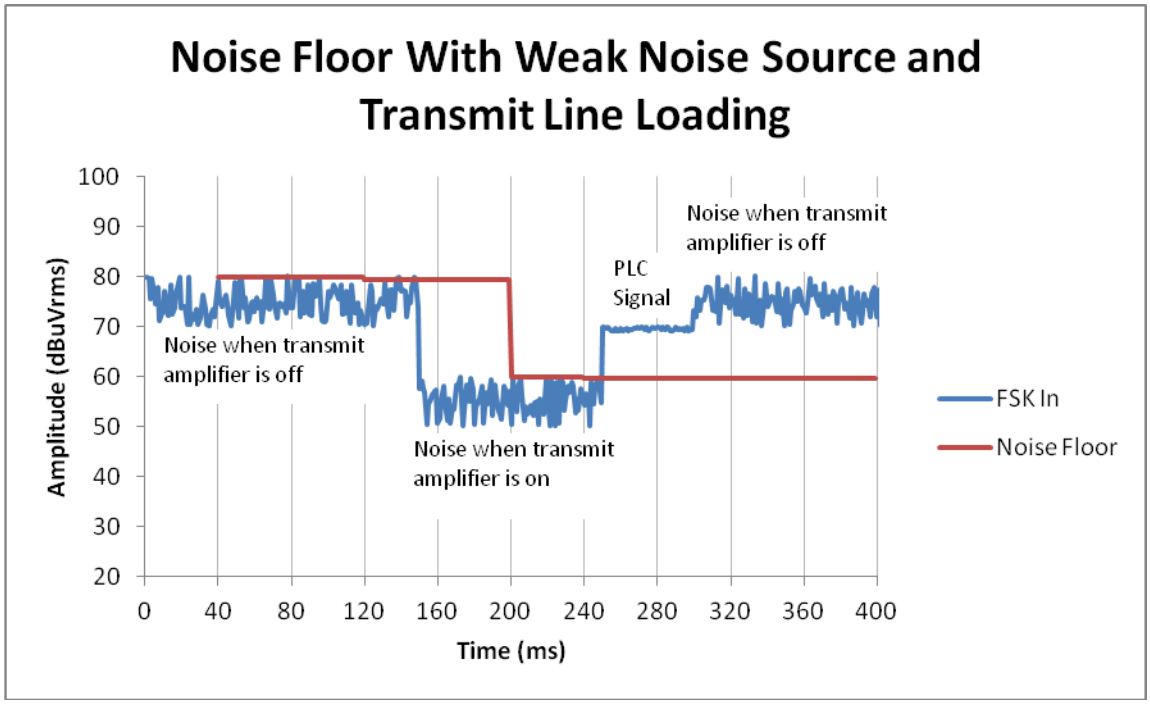
Figure 9. Frequency Hopping Algorithm



In some powerline environments, the noise near the receiver may be large, but is from a weak driving source (high source impedance). When the PLC transmitter is sending data to this receiver, it drives the powerline with a lower impedance, which will attenuate the noise. However, the receiver won't see the lower noise level because the PLC signal will hide the noise. For example, the noise level could normally be 80dB, but it drops to 60dB when the PLC transmitter is transmitting. If the PLC signal is 75 dB, the frequency hopping receiver will not detect it because its noise floor is 80dB. Since the SNR is actually 15dB (75dB PLC signal minus 60dB noise while transmitting), the receiver can correctly receive this signal when it's not frequency hopping. To handle this situation, before transmitting, the transmit amplification circuit is turned on, but the device doesn't transmit any PLC data. This will load the line, drop the noise floor, and give the receiver time to read the lower noise floor. The amount of time is set to 100ms, which is more than double the 40ms interval of the receiver's noise floor calculation. It needs to be more than double because the receiver could start sampling at any time and it won't accurately measure the noise floor if the loading was not on during the whole interval. Figure 10 shows the effect of the line loading at 150 ms, which causes the noise floor to drop to 60 dB.

The line loading feature is set by the EnableTxLineLoading() function and turned off by the DisableTxLineLoading() function.

Figure 10. Noise Floor Detection with Transmit Line Loading



Performance

The performance of the Network Protocol with the FSK Modulator, FSK Demodulator, and recommended coupling circuit is provided in the application note AN76458 “PSoC 5LP High Voltage (120 – 240 VAC) Powerline Communication”.

Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
1.0	First release.	First release.

© Cypress Semiconductor Corporation, 2013. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control, or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC® is a registered trademark, and PSoC Creator™ and Programmable System-on-Chip™ are trademarks of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.