

BC7215A Offline A/C Control Library V7.x

- ✓ Free of charge
- ✓ 100% offline
- ✓ IR command parsing
- ✓ Provided as C source
- ✓ Small footprint
- ✓ Low system requirements
- ✓ Supports both Celsius & Fahrenheit A/Cs

Index

Introduction.....	3
Introduction to AC Remote Control Protocols.....	4
How to Use the BC7215A A/C Control Library.....	4
Step 1: Sample the Original Remote Control Signal.....	4
Step 2: Initialize the A/C Control Library (Pairing).....	4
Step 3: Set Desired Temperature, Mode, and Fan Speed.....	4
Other: Power On/Off.....	5
Best Practices.....	5
System Requirements.....	5
A/C Control Library Details.....	5
Composition.....	5
Data Types.....	5
API Functions.....	6
1. Initialization Function.....	6
2. Initialization Function 2.....	7
3. A/C Setting Function.....	8
temp:.....	9
mode:.....	9
fan:.....	9
key:.....	9
4. A/C Power On/Off Functions.....	10
5. Command Parsing Function.....	11
6. Find Next Match Function.....	12
7. Predefined Data Packets.....	12
Get Predefined Data Count.....	13
Get Format Information Packet for Predefined Data.....	13
Get Raw Data Packet for Predefined Data.....	13
Get Predefined Data Name.....	13
8. Get Version Information Function.....	13
9. Check for Special Format Requirements (deprecated).....	13
10. Save Special Format Information (deprecated).....	14
11. Replace Base Data Packet.....	14
12. Get Base Data Packet.....	15
13. Get Base Format Packet.....	15
14. Get Special Format Information (deprecated).....	15

Introduction

The BC7215A Offline Air Conditioner Control Library (hereinafter referred to as the "BC7215A A/C Library") is an IR remote control library designed specifically for the BC7215A chip. When paired with the BC7215A, it enables offline, universal air conditioner control.

Unlike other libraries based on databases, this library utilizes a rule-based encoding approach. Instead of selecting a specific brand or model during setup, the system automatically identifies the encoding rules by decoding the infrared signal, thereby achieving universal AC control without relying on a database.

While there are thousands of AC brands and models on the market, they utilize only about a few hundred distinct remote control protocols. It is very common for different models—or even different brands from major manufacturers—to share the same IR protocol. Therefore, this rule-based universal solution offers the following advantages:

- **Completely Offline:** It does not rely on the internet or external databases and can run on low-specification microcontrollers.
- **Native Support for Future Models:** Air conditioner IR remote functions are generally fixed, meaning new products are highly likely to retain existing control protocols.
- **High Probability of Support for Niche Brands:** Smaller brands often adopt mature OEM solutions from major manufacturers rather than developing their own protocols, increasing the likelihood of compatibility.
- **This code library requires no internet connection or external resources.** It is provided as C source code (obfuscated) and has low processing power requirements, making it suitable for environments ranging from 8-bit MCUs to 64-bit Linux and Windows systems. The driver library is compact; Version 4.0 has a footprint of approximately 40KB when compiled on an ARM Cortex-M3 processor, requiring about 900 bytes of RAM (specifics vary by processor).

By fully leveraging the universal decoding capabilities of the BC7215A, this library implements a generic driver based on AC encoding rules rather than a waveform database. This allows it to remain completely offline while maintaining a small size.

The driver library provides four basic AC control functions:

- Temperature control
- Operation mode control
- Fan speed control
- Power On/Off control

Combined with the "Base Data Replacement" feature introduced in V3.0, users can achieve control over any specific AC setting when needed, such as swing direction (wind direction), eco mode, and more.

Since V5.0, the library offers function to "parse" the IR signal and read "Temperature", "Mode", "Fan Speed" and "Power Status" from it, it is the only product that has this ability on the market.

Introduction to AC Remote Control Protocols

In general, infrared (IR) remote controls for air conditioners fall into two categories:

1. **Fixed-Code Remote Control** This type is characterized by each button on the remote transmitting a fixed, unchanging IR code. The distinguishing feature of these remotes is the absence of a LCD screen. Essentially, they function the same way as standard IR remotes for audio/video (A/V) equipment. To control these, one simply needs to replicate their infrared signals. This can be achieved using the basic functions of the BC7215(A) without the need for a specialized driver library.
2. **Variable-Code Remote Control** This is the most common type. These remotes are characterized by long signal lengths, where each frame of the signal contains the complete status information for the air conditioner, such as the set temperature, operating mode, and fan speed. The distinguishing feature is simple: these remotes always include a LCD screen.

With this type of remote, the IR code transmitted by pressing the exact same button will differ depending on the current settings of the device (e.g., pressing "Temp Up" sends a different code if the target is 24°C vs 25°C). Furthermore, encoding methods vary significantly across different manufacturers and models. Consequently, generating these complex signals requires the assistance of this library.

How to Use the BC7215A A/C Control Library

Using this driver library is straightforward. You can achieve control over any air conditioner in just three steps.

Step 1: Sample the Original Remote Control Signal

Use the BC7215A's decoding function to receive and decode a specific signal from the target air conditioner's remote (specifically: Cool Mode, 25°C). This sampled data allows the driver library to analyze and identify the signal format of the original remote.

Note:

Do not use a "Universal Remote" as the signal source. These remotes typically transmit a sequence of multiple signals using different protocols for every button press to ensure broad compatibility. Since signal capture usually records only the first infrared signal detected, the BC7215A may capture an incorrect signal if the effective protocol is not the first one in the sequence.

Step 2: Initialize the A/C Control Library (Pairing)

Using the decoded data as a parameter, call the driver library's initialization function. This enables the A/C Control Library to recognize the specific encoding format of the target air conditioner.

Step 3: Set Desired Temperature, Mode, and Fan Speed

Use the control library's setup function to configure the required parameters. The function will return a new data packet. Simply transmit this data packet via the BC7215A to control the air conditioner.

Other: Power On/Off

The driver library also provides functions to toggle the air conditioner's power on and off. Please refer to the sections below for details.

Best Practices

In practical applications, the sampled data can be saved (e.g., in non-volatile memory). Upon system startup, initialization can be performed directly using this stored data. This eliminates the need for the user to repeat the sampling process every time, ensuring the system is "ready-to-use" immediately after powering on.

System Requirements

The BC7215A A/C Control Library is provided as C source code.

- **Compiler Standard:** Requires a compiler compatible with the C99 standard or higher.
- **Compatibility:** Theoretically, the library can be deployed on any system that supports the C programming language.
- **Resources:** The target system must have sufficient program memory (Flash/ROM) and RAM to accommodate the library.

A/C Control Library Details

Composition

The BC7215A A/C Control Library consists of two files:

`bc7215_ac_lib.h`

`bc7215_ac_lib.c`

`bc7215_ac_lib.c` contains the obfuscated C source code. Since this library is designed to be used in conjunction with the BC7215A chip, it utilizes data structures defined in the standard BC7215(A) driver library. Therefore, your system must also include:

`bc7215_lib.h`

And the associated files:

`bc7215_types.h`

`bc7215_lib_config.h`

Because the infrared codes for some air conditioners are quite long, it is recommended to set the `BC7215_MAX_RX_DATA_SIZE` value in the BC7215 driver library to 48 or higher. (Note: Larger values will consume more RAM).

Data Types

Starting from V3.0, the BC7215A A/C Control Library introduced a dedicated data format.

```

typedef struct {
    uint16_t    bitLen;    /**< Placeholder for compatibility with data packet format;
value is always 0 */

    union {

        uint8_t data[1];    /**< Start position of the data packet's data */

        struct {

            const bc7215FormatPkt_t*    fmt;    /**< Pointer to the format packet */

            const bc7215DataVarPkt_t*    datPkt; /**< Pointer to the data packet */

        } msg;    /**< New data structure containing two pointers */

    } body;    /**< New data format utilizes the data section of the original data packet
to store pointers to the format and data packets */

} bc7215CombinedMsg_t;

```

This data format is primarily used to pass infrared signal format information between the A/C Control Library and the user application. This format information is used to allow the BC7215A chip to generate infrared waveforms that are consistent with the original infrared signal.

The library itself does not need or process this format information.

Pre-V3.0: Format information was handled entirely by the user program.

V3.0: Added format information management functionality. It now supports rare air conditioner protocols where different commands use different infrared formats. The library will provide the corresponding format data for commands requiring special formats.

The new data structure is designed to be compatible with the data packet `bc7215DataVarPkt_t`. Wherever a data packet pointer was passed in previous library functions, this new data structure can now be used.

Detection Logic: The user simply needs to check the `bitLen` of the data packet.

If `bitLen == 0`: Since a normal data packet cannot have a bit length of 0, this indicates that the structure is actually a `combinedMsg_t`.

The user can then type-cast the pointer to retrieve the passed format and data packets.

API Functions

To control an air conditioner, add the library files to your C project and include the header file for the BC7215A A/C Control Library in your source code. You can then call the following API functions.

Please note that several functions are categorized into distinct versions for Celsius and Fahrenheit systems. You must select the function variant that corresponds to your specific air conditioner hardware.

1. Initialization Function

```

bool bc7215_ac_init(uint8_t status, const bc7215DataVarPkt_t* dataPktCool25C);

bool bc7215_ac_init_f(uint8_t status, const bc7215DataVarPkt_t* dataPktCool78F);

```

This function initializes the driver library. There are 2 variants of the function: `bc7215_ac_init()` and `bc7215_ac_init_f()`. These are used for air conditioners operating in Celsius and Fahrenheit systems, respectively.

Important: You must call the specific function that matches the unit system of your hardware. Failure to do so may result in unexpected temperature control behavior, even if the initialization returns a successful status.

The function accepts two input parameters:

- The status byte from the raw data packet, obtained when the BC7215A decodes the captured signal from the target A/C remote.
- A pointer to the corresponding raw data packet.

Returns: A bool value. true indicates successful initialization; false indicates failure.

Important: The raw data packet used for initialization must correspond to the air conditioner set to "Cooling Mode, 25°C" or "Cooling Mode, 78°F". If the input data does not match this specific state, initialization will fail.

Tip: First set the A/C remote to "Cooling Mode, 25°C", then press the "Fan Speed" button to capture the signal. This ensures that the operating mode and temperature settings remain unchanged during the transmission.

V3.0 Update: Starting from V3.0, the second parameter supports the `combinedMsg_t` format. However, it must still be passed as type `const bc7215DataVarPkt_t*`. The driver library internally checks the `bitLen` value to determine the actual type:

If `bitLen == 0`, the library interprets the input as `combinedMsg_t`.

It then saves a copy of the contained format information internally within the driver.

This format information can subsequently be retrieved using the `bc7215_ac_get_base_fmt()` function.

2. Initialization Function 2

```
bool bc7215_ac_init2(uint8_t msgCnt, const bc7215CombinedMsg_t msgs[], uint8_t segGap);  
bool bc7215_ac_init2_f(uint8_t msgCnt, const bc7215CombinedMsg_t msgs[], uint8_t segGap);
```

This function is designed to handle remote control protocols where the infrared signal is split into multiple segments. It also has Celsius and Fahrenheit 2 variants.

The BC7215A chip identifies the end of an infrared signal based on the gap between signals (currently set to 36ms). While the vast majority of air conditioning IR signals consist of a single continuous stream, a small number of protocols split a single control command into two or more segments with intervals exceeding 36ms.

The V4.0 library automatically detects this situation and alerts the user via the `bc7215_ac_need_extra_sample()` function, indicating that multi-segment sampling is required to complete initialization correctly. The input data must still correspond to "Cooling Mode, 25°C" (or "Cooling Mode, 78°F for Fahrenheit).

Returns: A bool value. The meaning is the same as `bc7215_ac_init()`.

Parameters:

- msgCnt: The number of IR signal segments (Range: 1-4).
- msgs: A pointer to an array of type bc7215CombinedMsg_t. The used elements in the array must contain pointers to the data and format information for each IR signal segment. For example, if msgCnt is 2, the first two elements of the array must contain the pointers for the two IR signal segments; subsequent elements are ignored.
- segGap: The interval between IR signal segments in milliseconds (ms). The maximum interval the BC7215A can generate is 72ms; values exceeding this will be capped at 72ms. If set to 0, the most common value of 60ms will be used.

Note: This timing is usually not critical; air conditioners can typically receive signals correctly across a wide range of intervals.

Important Usage Note: Since this function does not accept status information as an input parameter, the user must check the status word of the received raw data packet before passing the data.

If the b6:REV (Invert) flag is set to 1, the user must manually invert the data to restore it before passing it to this function. (The standard bc7215_ac_init() function handles this check internally, as it includes the status byte in its parameters).

If the input segment count is 1, this function is equivalent to bc7215_ac_init().

3. A/C Setting Function

```
const bc7215DataVarPkt_t* bc7215_ac_set(int8_t Ctemp, int8_t mode, int8_t fan, int8_t key);
```

```
const bc7215DataVarPkt_t* bc7215_ac_set_f(int8_t Ftemp, int8_t mode, int8_t fan, int8_t key);
```

After successful initialization, use these functions to configure the air conditioner's state. There are also separate functions for Celsius and Fahrenheit.

The function accepts four input parameters. The first three correspond to temperature, operating mode, and fan speed. The fourth parameter is the key type.

Understanding the 'Key' Parameter: In the IR protocols of some air conditioners, the encoded signal includes information about the specific button pressed.

- Example: Reaching 25°C by pressing "Temp -" (down from 26°C) may generate a different IR code than reaching 25°C by pressing "Temp +" (up from 24°C), even though the final state (25°C) is identical.
- This behavior occurs only in specific AC models.
- Recommendation: If you are unsure whether your specific application requires this, it is generally safe to set this parameter to "No Change" or "Temp +".

Parameter Validity: All input parameters are 8-bit signed integers (int8_t). Each parameter has a defined valid range. Any input value falling outside its specific range will be treated as "keep current setting" (i.e., that specific setting will remain unchanged).

temp:

For Celsius: Value Range: 0–14, corresponding to temperature values of 16°C–30°C.

For Fahrenheit: Value Range: 0–28, corresponding to temperature values of 60°F–88°F.

Any parameter value outside this range is treated as "No Change" to the existing setting.

mode:

Value Range: 0-4,

0 - (Auto)

1 - (Cool)

2 - (Heat)

3 - (Dry)

4 - (Fan)

Value outside this range is treated as "No Change" to the existing setting.

fan:

Value Range: 0-3,

0 - (Auto)

1 - (Low)

2 - (Med)

3 - (High)

Value outside this range is treated as "No Change" to the existing setting.

key:

Value Range: 0-3,

0 - Temperature +

1 - Temperature -

2 - Mode

3 - Fan Speed

Value outside this range is treated as "No Change" to the existing setting.

Return Value and Transmission

The return value of this function is a pointer to a BC7215(A) data packet. This packet contains the data required for the user's settings, encoded according to the target air conditioner's format. By sending this packet to the air conditioner using the Transmit Command (F5 02), control is achieved.

For details on the Transmit Command, please refer to the BC7215(A) Datasheet and the BC7215(A) Driver Library Manual.

Special Case (Format Change):

If the bitLen of the data packet pointed to by the return result is 0, this indicates that the output is actually a combinedMsg_t* pointer containing a pointer to a format information packet. This signifies that the specific command (e.g., a Mode Change operation) uses a signal format different from the standard signal and requires a dedicated format packet. In this case, use the format and data packet pointers contained within the combinedMsg_t structure to retrieve the corresponding data.

Notes:

1. **Validity of Settings:** The BC7215A A/C Control Library is a rule-based library. It is responsible only for generating data packets that comply with encoding rules; it does not validate whether a specific setting combination is logically valid for the target device. Many combinations of settings may be physically impossible for actual air conditioners, and limitations vary by model. For example, most models ignore temperature settings in "Fan" mode, and some models may lack a "Heat" mode entirely. When using this library, adhere to the actual limitations of the target model. If settings exceed the allowable range of the specific machine, the air conditioner's response will be unpredictable.
2. **Flow Control for Long Data:** When the data length exceeds 16 bytes (the size of the BC7215(A) internal buffer), you must use flow control based on the BUSY signal. Failure to do so will result in corrupted infrared signal transmission, rendering it unrecognizable to the air conditioner. For details on flow control, please refer to the BC7215(A) Datasheet.
3. **Format Information Persistence:** The format information packet contains the modulation format of the infrared signal. Users must load this into the BC7215A chip after it enters Transmit Mode. Once loaded, the format information persists within the chip as long as the BC7215A does not exit Transmit Mode or lose power; it does not need to be reloaded before every transmission.

4. A/C Power On/Off Functions

```
const bc7215DataVarPkt_t* bc7215_ac_on(void);  
const bc7215DataVarPkt_t* bc7215_ac_off(void);
```

These two functions are used to generate commands for controlling the A/C power on/off state. There's no C/F difference for this function. Unless explicitly stated otherwise, all subsequent functions are unit-independent and do not distinguish between Celsius and Fahrenheit systems.

Power On Logic: Typically, air conditioners do not require a dedicated "Power On" command. If the unit is in the "Off" state, sending a status setting command (e.g., changing mode or temperature) will automatically turn it on.

For such models, calling bc7215_ac_on() will return NULL.

However, a few specific models may require a dedicated command to power on; in such cases, the function returns a valid pointer to the data packet containing the "Power On" command.

Power Off Logic: The bc7215_ac_off() function returns a pointer to the data packet containing the shutdown instruction. Transmitting this packet will turn off the air conditioner.

Special Case (Format Change): If the bitLen of the data packet pointed to by the return result is 0, this indicates that the output is actually a combinedMsg_t* pointer. This implies that this specific

command (Power On or Off) utilizes an infrared signal format different from the standard signal and requires a dedicated data packet.

Action Required: Use the format packet pointer contained within the combinedMsg_t structure to retrieve the format information specific to this command, and load it into the BC7215A chip prior to transmission.

Important Note: When resuming transmission of other standard data packets afterwards, you must restore the format information within the chip back to the standard format.

5. Command Parsing Function

```
bool bc7215_ac_parse(int8_t* Ctemp, int8_t* mode, int8_t* fan, int8_t* power);  
bool bc7215_ac_parse_f(int8_t* Ftemp, int8_t* mode, int8_t* fan, int8_t* power);
```

This function is used to parse and extract temperature, mode, fan speed, and power status parameters from an infrared signal. Correct C or F version must be used according to the A/C hardware.

Parameters and Output: The function accepts four input parameters, all of which are pointers to 8-bit signed integers. The parsing results will be written to the variables pointed to by these pointers. The meanings of the output values are identical to those used in the A/C setting function, bc7215_ac_set().

The output power value has three valid states:

0: Off

1: On

2: Toggle (This indicates that in some remote protocols, the power command flips the power state, meaning the command data for turning "On" and "Off" is identical.)

Any value other than these indicates that the specific item could not be parsed.

Return Value: Returns a bool value.

true: Parsing was successful.

false: Parsing failed (e.g., due to incorrect input data format).

Prerequisites: This function can only be used after the A/C Control Library has been correctly initialized.

Usage Requirement: Before calling this function, the data to be parsed must be loaded using the bc7215_ac_replace_base() function. The user must ensure that the loaded data belongs to the same protocol format as the data used during initialization; otherwise, parsing will fail.

Limitations (Experimental Version)

This function is currently an experimental version and has the following limitations:

- Format Constraints: Currently, it only supports parsing data that shares the same format as the base data packet. It does not yet support parsing of pre-defined data protocols or instructions with special formats. For example, if a protocol uses a special format for "Mode" settings, instructions generated by pressing the "Mode" button cannot be parsed.

- **Discrepancies in Parameter Parsing:** It is not guaranteed that every parsed result will match the display on the physical remote control. Discrepancies may occur in the following scenarios:

Temperature	Context: In certain modes like "Dehumidify" (Dry) or "Fan," temperature settings may not be relevant. Manufacturers might program the remote to send a specific fixed temperature or a special out-of-range value. Whether the remote displays a temperature or not depends on the manufacturer. Result: In these cases, temperature parsing will fail.
Mode	Context: Apart from "Cool" and "Heat," other modes might not have dedicated setting values but are instead defined by specific combinations of settings. For example, a manufacturer might define "Dehumidify" simply as "16°C + Cool + Low Fan." Result: The function will parse this as "Cool" mode, which differs from the "Dehumidify" mode shown on the remote.
Fan Speed	Context: Manufacturers often make minor modifications to standard protocols for specific models. The most common modification involves preset fan speed values. Result: This can occasionally lead to incorrect parsing results or parsing failures.
Power	Context: Power On/Off commands often employ special instruction formats or content. Result: This may lead to parsing failures. Additionally, commands intended to explicitly turn power On or Off might only be parsed as "Toggle," making it impossible to determine the absolute power state.

- Executing `bc7215_ac_replace_base()` modifies the base data of the BC7215A A/C Control Library. Consequently, all subsequent outputs from `bc7215_ac_set()` will be based on the most recently loaded data packet.

6. Find Next Match Function

```
bool bc7215_ac_find_next(void);
```

During initialization, the code library automatically identifies the encoding protocol used by the air conditioner based on the sampled data. However, given the vast number of A/C brands and models on the market, some encoding protocols are very similar yet slightly different. Occasionally, after initialization, the air conditioner's actual behavior may not match the settings.

If this occurs, you can try calling this function to search the library for other protocols that may match the target air conditioner. This function takes no input parameters and returns a bool value, which carries the same meaning as the return value of the initialization function.

- If the return value is true: The library has successfully re-initialized using a new protocol. You can then attempt to control the A/C to verify if it operates correctly. If issues persist, you can call this function again.
- If the return value is false: There are no further matching encoding protocols available.

Usage Constraints: This function can only be invoked after a successful initialization. If you wish to revert to the original protocol used immediately after the initial setup, you must call the initialization function again.

7. Predefined Data Packets

There are a very small number of A/C remote protocols that cannot be directly decoded by the BC7215A chip. Users can convert these using online conversion tools. The BC7215A A/C Control Library includes pre-converted, preset data for direct user access.

If you find that the BC7215A fails to complete data sampling, or if the sampled signal does not function correctly, you can attempt to use this predefined data to see if it resolves the issue.

The primary functions related to predefined data are as follows:

Get Predefined Data Count

```
uint8_t bc7215_ac_predefined_cnt(void);
```

Return Value: The quantity of built-in predefined data sets available in the library.

Get Format Information Packet for Predefined Data

```
const bc7215FormatPkt_t* bc7215_ac_predefined_fmt(uint8_t index);
```

Input Parameter: The index of the predefined data. The range starts from 0 and must be less than the return value of the `bc7215_ac_predefined_cnt()` function.

Get Raw Data Packet for Predefined Data

```
const bc7215DataVarPkt_t* bc7215_ac_predefined_data(uint8_t index);
```

```
const bc7215DataVarPkt_t* bc7215_ac_predefined_data_f(uint8_t index);
```

This function has dedicated versions for Celsius and Fahrenheit.

Input Parameter: Same as the function above (the index).

Return Value: A pointer to the data packet. Usage: The user can directly use this returned pointer to initialize the BC7215A A/C Control Library.

Get Predefined Data Name

```
const char* bc7215_ac_predefined_name(uint8_t index);
```

Input Parameter: The index of the predefined data.

Return Value: A string containing the mnemonic name of the preset data, which helps the user distinguish between different data sets.

8. Get Version Information Function

```
const char* bc7215_ac_get_ver(void);
```

This function requires no input parameters. It returns a string containing the current version information of the code library.

9. Check for Special Format Requirements (deprecated)

This function is no more necessary since V7.0, but will remain accessible for compatibility reason currently, shall be removed in future versions.

After successful initialization, you can call the following function:

```
uint8_t bc7215_ac_need_extra_sample(void);
```

This function returns an 8-bit integer indicating whether the currently initialized A/C protocol requires special formats for specific commands (that differ from the standard command format).

Return Values:

0. No special format required.
1. Temperature setting commands require a special format.
2. Mode setting commands require a special format.
3. Fan speed setting commands require a special format.
4. The protocol uses multi-segment IR signals and must be initialized using the `bc7215_ac_init2()` function.

10. Save Special Format Information (deprecated)

This function is no more necessary since V7.0, but will remain accessible for compatibility reason currently, shall be removed in future versions.

If `bc7215_ac_need_extra_sample()` returns a non-zero result, the user must perform an additional sampling of the command requiring the special format so that the driver library can learn this format information.

The input parameters are similar to the initialization function, but this function only accepts a pointer of type `combinedMsg_t`. Upon execution, a copy of the special format information and data packet will be saved within the driver library.

```
bool bc7215_ac_save_2nd_base(uint8_t status, const combinedMsg_t* message);
```

Returns: true if saved successfully, false otherwise.

11. Replace Base Data Packet

This function replaces the base data used to generate air conditioning data packets.

```
bool bc7215_ac_replace_base(uint8_t status, const bc7215DataVarPkt_t* altDataPkt);
```

Primary Uses:

1. **Command Parsing:** Before calling the command parsing function, this function must be called to replace the base data with the data to be parsed.
2. **Controlling Arbitrary A/C Functions:** The BC7215A A/C Control Library natively provides four basic control functions: Temperature, Mode, Fan Speed, and Power. Typically, air conditioners possess many other features, such as Swing (Wind Direction), Eco Mode, etc. These functions correspond to specific data bits within the A/C command. By replacing the data packet used during initialization with a packet containing the desired control information, you can achieve control over these specific functions. Example: Suppose the A/C status during library initialization was "Fixed Wind Direction." If we wish to set the A/C to "Auto Swing," we simply use this function to replace the base data packet with a packet captured in the "Auto Swing" state. This enables Swing functionality. Switching between these two base packets allows for arbitrary control of the Swing feature.

Note: When replacing the base packet, the Temperature, Mode, and Fan settings within the new packet can be in any state (they are ignored as the library recalculates them).

V5.2 Multi-Segment Support: The function has two input parameters: the status word of the data packet and the pointer to the data packet.

- Single-segment protocols: Call the function directly as described above.
- Multi-segment protocols (V5.2+): This function can now accept multi-segment A/C data. When doing so:

The first parameter (status) passes the segment count (MsgCnt).

The second parameter passes a pointer to the bc7215CombinedMsg_t array containing the multi-segment data.

Since the function signature remains unchanged, you must use type casting in your code to convert the bc7215CombinedMsg_t array pointer to a data packet pointer type when passing multi-segment data:

```
bc7215_ac_replace_base(uint8_t MsgCnt, (const bc7215DataVarPkt_t*)MsgArray[]);
```

Note: When used to replace the base data packet, the format packet portion within the bc7215CombinedMsg_t type is not used and can be any value.

Important: The replacement data packet must follow the same format as the one used during initialization (i.e., originating from the same remote control). Since the BC7215A chip can receive signals from any remote control, to prevent signals of other formats from being used to replace the base data packet, it is recommended that users compare the length of the input data packet with the initialization data packet before calling this function. Only data packets of the same length should be used for replacement.

12. Get Base Data Packet

```
const bc7215DataVarPkt_t* bc7215_ac_get_base_data(void);
```

Calling this function retrieves the base data packet currently being used (the one set during initialization or updated via replacement).

13. Get Base Format Packet

```
const bc7215FormatPkt_t* bc7215_ac_get_base_fmt(void);
```

If the initialization utilized a format containing a format packet (via combinedMsg_t), calling this function returns that format packet. If initialization did not use specific format information, the return result is undefined.

14. Get Special Format Information (deprecated)

This function is no more necessary since V7.0, but will remain accessible for compatibility reason currently, shall be removed in future versions.

```
bc7215CombinedMsg_t bc7215_ac_get_2nd_base(void);
```

For protocols that require extra key sampling, this function can be used after initialization to retrieve the special format information previously saved by the bc7215_ac_save_2nd_base() function. The return value is a bc7215CombinedMsg_t.