

BC7215AC Arduino
Universal
Air Conditioner Remote Control
Library

V7.x

Index

Air Conditioner Remote Control Protocol Overview.....	3
BC7215AC Arduino Air Conditioner Remote Control Library Usage Instructions.....	4
Step 1: Pairing with A/C.....	4
Step 2: Control A/C, or decode IR signal.....	4
Best Practice.....	4
System Requirements.....	4
Detailed Description of the Air Conditioner Remote Control Library.....	4
Library Composition.....	4
Hardware Connections.....	5
Public Properties.....	5
Functions.....	6
1. Start/Stop Sampling Functions.....	6
2. Sampling Status Query Functions.....	6
3. Initialization Functions.....	6
4. Air Conditioner Setting Functions.....	6
Temperature (temp):.....	6
Mode (mode):.....	7
Fan Speed (fan):.....	7
Key (key):.....	7
5. Air Conditioner Power On/Off Functions.....	7
6. IR Command Parse Funciont.....	7
7. Find Next Matching Function.....	8
8. Predefined Protocols.....	8
Acquire Number of Predefined Protocols.....	8
Acquire Name of a Predefined Protocol.....	8
Initialize Using a Predefined Protocol.....	8
9. Acquire Library Version Information.....	8
10. Retrieve Base Data Packet and Base Format Packet.....	8
11. Set Temperature Mode.....	8
12. Query The Current Temperature Mode.....	9
13. Query BC7215A Chip Status.....	9
Programming Flowcharts.....	10
Issue Report.....	12

Introduction

The BC7215AC Arduino Universal Air Conditioner Remote Control Library (hereinafter referred to as the AC Remote Control Library) is a dedicated code library designed for Arduino systems. It is derived from the C-language implementation of the BC7215A offline air conditioner remote control code library and has been adapted to accommodate the specific characteristics of Arduino platforms. This library requires the BC7215A chip for operation; consequently, it incorporates both the BC7215 chip driver library and the universal air conditioner remote control functionality, requiring only a single installation by the user. The current version encompasses the majority of air conditioner brands and models, with continuous iterative enhancements.

In contrast to libraries reliant on sample databases, this library employs an encoding rule-based methodology. Configuration is achieved not through brand or model selection but via automatic protocol identification from decoded infrared signals, facilitating a "**Pairing-And-Control**" two-step setup, users don't even need to know the A/C's brand or model. This rule-based universal remote control approach provides the following advantages:

- Independent of networks and databases; fully offline operation suitable for resource-constrained microcontrollers.
- Inherent compatibility with future models, as air conditioner infrared remote functions are largely standardized, and new products typically adopt established protocols.
- Enhanced support for less common brands, which frequently utilize mature OEM solutions from established vendors.

This library operates without internet connectivity and depends solely on the BC7215A chip, with no additional external dependencies. It is compatible with any Arduino system possessing adequate program memory and RAM. Accompanying examples demonstrate usage on ESP8266, ESP32 and Nano33 IoT modules.

Leveraging the BC7215A's universal decoding capabilities, the library implements a protocol-rule-driven general-purpose driver, as opposed to a waveform database. This design ensures compactness and offline functionality. The library supports four fundamental air conditioner control operations:

- Temperature adjustment
- Operating mode selection
- Fan speed control
- Power on/off management

Meanwhile, it can also parse(decode) the IR signal, extract temperature, working mode, fan speed and power status from the remote controller.

Air Conditioner Remote Control Protocol Overview

Infrared remote controls for air conditioners are categorized into two primary types. Fixed-code remotes transmit invariant infrared codes per button and are identifiable by the absence of a LCD display on the remote. These resemble standard audiovisual equipment remotes and can be replicated using the BC7215(A)'s core functions without a specialized library; refer to the "learning remote control" example in the BC7215 Arduino driver library.

The predominant type involves variable-code remotes, featuring extended signal lengths that encapsulate comprehensive control data, including set temperature, operating mode, and fan speed. These are distinguished by the presence of a LCD display. Codes from identical buttons vary with the device's state, and encoding differs by manufacturer and model. Signal generation for such remotes necessitates this AC Remote Control Library.

BC7215AC Arduino Air Conditioner Remote Control Library Usage Instructions

Note: Given the library's dependence on the BC7215A chip, users are advised to familiarize themselves with the chip's operational principles, data formats, and driver functions prior to proceeding. This knowledge facilitates comprehension. Relevant documentation is available in the library's documentation directory.

The library's utilization is straightforward, comprising 2 essential steps for controlling any air conditioner.

Step 1: Pairing with A/C

To capture a designated signal from the target air conditioner's remote: cooling mode at 25°C (or 78°F), the sampled data will be analysed and the protocol is identified, then they are paired.

Note: Refrain from using universal remotes as signal sources, as they often emit multiple protocols sequentially per button press, this will make the pairing fail.

Step 2: Control A/C, or decode IR signal

Once paired, you can control the A/C or decode the IR signal to get temperature etc.

Best Practice

Store the pairing data for next use. so upon system startup, it will be automatically paired and enabling immediate operability. Consult the example programs for implementation details.

System Requirements

The library is applicable to any Arduino system with sufficient program storage and RAM capacity. Demonstration programs utilize ESP8266, ESP32 and Nano33 platforms.

Detailed Description of the Air Conditioner Remote Control Library

Library Composition

The BC7215AC library comprises two distinct driver components: the BC7215(A) chip driver library for fundamental chip control, and the specialized air conditioner remote control library. The chip driver library includes four demonstration programs:

- Arbitrary remote control decoding
- Learning-type remote control

- Dual infrared remote switch
- Infrared data transmission

Air conditioner applications mandate the BC7215A variant, as the standard BC7215 is unsuitable for air conditioner signal decoding. The air conditioner remote control library provides eight demonstration programs:

- **ESP8266 non-blocking version** (software serial communication with BC7215A; Arduino IDE serial monitor for interaction; non-blocking design)
- **ESP32 serial monitor English version** (port of ESP8266 demonstration)
- **ESP32 serial monitor Chinese version** (port of ESP8266 demonstration)
- **ESP32 LCD English version** (utilizes LILYGO T-Display onboard buttons and LCD)
- **ESP32 LCD Chinese version** (utilizes LILYGO T-Display onboard buttons and LCD)
- **ESP32 MQTT English version** (extends LCD version with networking for MQTT-based control and status reporting)
- **ESP32 MQTT Chinese version** (extends LCD version with networking for MQTT-based control and status reporting)
- **Nano33 IoT English Serial Monitor version.**
- **ESP32 Home Assistant Version** (Based on ESP32 MQTT example, automatically integrate with HA)
- **ESP8266 Home Assistant Version** (Home Assistant version with no onboard LCD display)

Hardware Connections

4 wires are required for the connection: 2 for UART, and 2 for GPIOs. The BC7215A chip serves as the core component for air conditioner control, interfacing with the Arduino via serial communication. Two auxiliary signals are required: MOD (Arduino output for transmit/receive mode selection) and BUSY (BC7215A output for serial data transfer control).

Public Properties

In a BC7215AC library class subject, there are several public properties can be accessed:

```
bool    initOK;           -- bool type, indicate whether the library has been paired

uint8_t sampleCount;      -- 8bit unsigned int, the segment count in a sampling process, maximum 4.

uint8_t sampleStatus[4];  -- 8bit unsigned int array, status for each segments in sampling.

bc7215DataMaxPkt_t  sampleData[4];      -- Sampling data for each segments.

bc7215FormatPkt_t   sampleFormat[4];    -- Sampling format for each segments.
```

Functions

The following delineates the interface functions of the BC7215AC air conditioner remote control library. For details on the chip driver library functions, consult the BC7215 Arduino driver library manual.

1. Start/Stop Sampling Functions

```
startCapture();
```

```
stopCapture();
```

These functions initiate and terminate the BC7215A chip's receive mode for sampling air conditioner remote signals prior to library initialization.

2. Sampling Status Query Functions

```
bool signalCaptured();
```

These functions ascertain whether a complete signal has been sampled.

Returns true upon completion.

3. Initialization Functions

```
bool init();
```

```
bool init(const bc7215DataMaxPkt_t& data, const bc7215FormatPkt_t& format);
```

These functions initialize the library using sampled data to complete the pairing process. The simplest form employs the internal buffer, suitable for being called right after a capture has been done. The 2nd accepts explicit data and format inputs, suitable for restored pairing data.

Returns true on successful initialization.

For Celsius A/Cs, the input data must originate from cooling mode at 25°C, or 78°F for Fahrenheit system, otherwise the pairing will fail. It's recommended to set the remote to "Cooling Mode" and specified temperature, then press "Fan Speed" button.

4. Air Conditioner Setting Functions

```
const bc7215DataVarPkt_t* setTo(int temp, int mode = -1, int fan = -1, int key = 0);
```

After initialization, this function configures the air conditioner state. Parameters include temperature, mode, fan speed, and key (for protocols incorporating button-specific encoding).

Default values allows sequential omission. Out-of-range values imply no change for the respective parameter.

Calling this function triggers immediate infrared transmission; returns a pointer to the transmitted data.

Temperature (temp):

Range: For Celsius system, 16-30 (corresponding to 16°C-30°C); For Fahrenheit, it should be ranged from 60 to 88. Exceeding range means no change to temperature.

Mode (mode):

Range: 0-4 . Exceeding range: no change.

- 0 - Auto
- 1 - Cool
- 2 - Heat
- 3 - Dry (Dehydrate)
- 4 - Fan

Fan Speed (fan):

Range: 0-3 . Exceeding range: no change.

- 0 - Auto
- 1 - Low
- 2 - Med
- 3 - High

Key (key):

Range: 0-3. Exceeding range: no change.

- 0 - Temperature +
- 1 - Temperature -
- 2 - Mode
- 3 - Fan

For most A/Cs this parameter can be ignored, but for some brands this parameter is critical, for example if you want to change the working mode, you must set this item to "2-Mode" to make the command effective.

Note: The library generates protocol-compliant packets without validating setting validity. Air conditioners impose model-specific constraints (e.g., temperature irrelevance in fan mode, or some models are cooling only without heating mode). Adhere to target device limitations; invalid settings may yield unpredictable behavior.

5. Air Conditioner Power On/Off Functions

```
const bc7215DataVarPkt_t* on(void);  
  
const bc7215DataVarPkt_t* off(void);
```

These functions manage power state. Invocation transmits the corresponding signal; returns pointer to data to be sent.

Most models activate upon state-setting commands in off state, reverting to prior or initialization state. Select models require dedicated ON command, followed by setTo() for state configuration.

6. IR Command Parse Function

```
bool parse(int temp, int mode, int fan, int power);
```

Parse the just captured IR signal, the result will be write into the 4 parameters. The meaning and ranges for the parameters are the same with the setTo() function, an out of range return value means unsuccessful parsing for that item, while the if the return value is false, means the whole signal was not able to be parsed.

7. Find Next Matching Function

```
bool matchNext();
```

Initialization automatically matches protocols; however, similar variants may exist, causing discrepancies. This function searches for alternative matches. Returns true on success; test functionality. Repeat until false.

Available only after a successful pairing; call init() to start over.

8. Predefined Protocols

Certain protocols elude direct BC7215A decoding, necessitating waveform capture and online conversion. The library includes pre-converted data. Employ if standard sampling/initialization fails.

Associated functions:

Acquire Number of Predefined Protocols

```
uint8_t cntPredef();
```

Returns the count of embedded predefined protocols.

Acquire Name of a Predefined Protocol

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

Returns mnemonic name for specified index.

Initialize Using a Predefined Protocol

```
bool initPredef(uint8_t index);
```

Initializes with embedded protocol by index.

9. Acquire Library Version Information

```
const char* getLibVer();
```

Returns a string containing the library version.

10. Retrieve Base Data Packet and Base Format Packet

```
const bc7215DataVarPkt_t* getDataPkt();
```

```
const bc7215FormatPkt_t* getFormatPkt();
```

Obtain current initialization data for storage and future use.

11. Set Temperature Mode

```
void setFahrenheit();
```



```
void    setCelsius();
```

Before any operation, the temperature mode must be selected, otherwise the pairing could fail, or the temperature control could behave unexpected.

Celsius is the default mode.

12. Query The Current Temperature Mode

```
bool    isCelsius();
```

If current mode is Celsius, will return 'true', otherwise will return 'false'.

13. Query BC7215A Chip Status

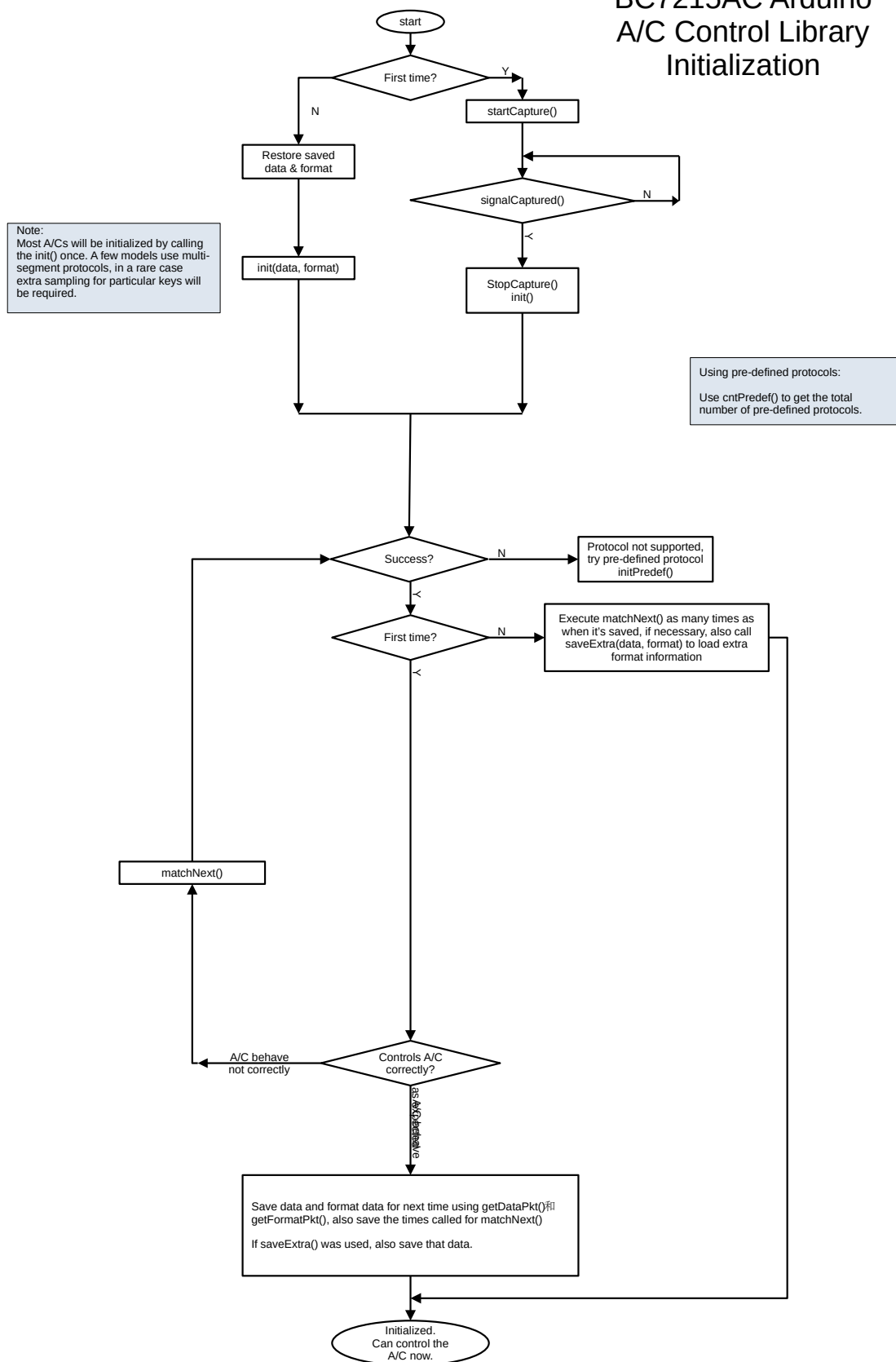
Please note this is a function from BC7215 Driver Library, not the A/C Control Library.

```
bool    isBusy();
```

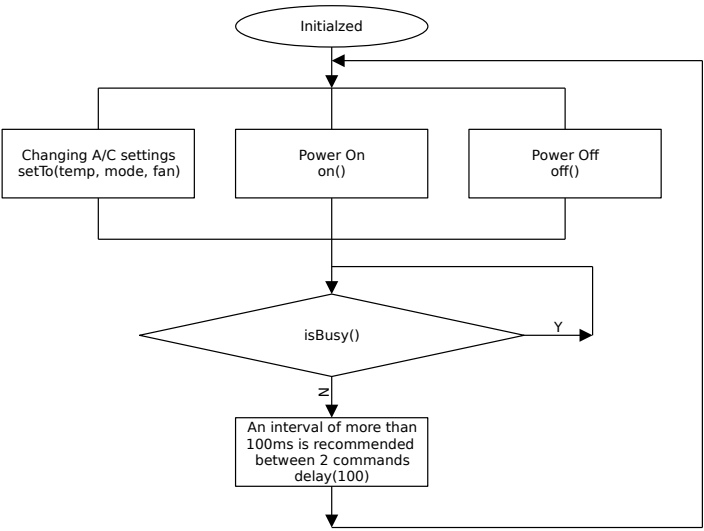
Returns true during signal transmission/reception. Post-transmission, await completion and >100ms interval before subsequent signals to ensure proper reception.

Programming Flowcharts

BC7215AC Arduino A/C Control Library Initialization



A/C Control Flowchart



Issue Report

If you encountered an unsupported A/C model, or though the pairing was successful, but the A/C unit didn't behave as expected, you are welcome to submit an issue report, the report form is "AC_Lib_Report-空调码库问题上报.pdf", it is located in the same folder as this document. Usually the issue will be fixed in 1 week and you'll get the upgraded version.

BC7215A 离线空调码库问题报告单

BC7215A Offline A/C Library Issue Report

如果您在使用 BC7215A 离线空调码库的过程中遇到了码库初始化失败或能初始化但控制空调失败的情况，请详细填写此表格并将填写后的表格和遇到问题的空调遥控器照片，以及空调机照片，一并作为邮件附件发送至邮件：ac-lib-issue@bitcode.com.cn，您将收到确认邮件，问题经技术人员确认后，您将收到报告单号，以及问题解决的解决方案。

If you experience library initialization failure or control failure (after successful initialization) when using the BC7215A offline AC control library, please complete this form. Email the completed form as an attachment to ac-lib-issue@bitcode.com.cn, including photos of the AC remote control and the AC unit. You will receive an initial confirmation email. After our technical team validates the issue, we will provide you with a ticket number and subsequent solution.

带*项目为必填
Fields marked with an asterisk (*) are required.

1. 用户信息 User information	*姓名 Name	
	公司名称 Company name	
	*联系电话 Contact email	
2. 空调信息 A/C information	*品牌 Brand	
	型号 Model	
	*生产年份 Year of manufacture	
3. 使用信息	*使用码库版本 AC Lib Version	

Usage information	*是否使用了官方演示板和演示软件测试 Tested with official demo board and demo software	<input type="radio"/> Yes	<input type="radio"/> No
4. 问题详情 Issue Details	*是否已尝试了所有的预定义数据 Have already tried all the pre-defined data	<input type="radio"/> Yes	<input type="radio"/> No
	*是否能正常解码红外信号 IR signal can be decoded	<input type="radio"/> Yes	<input type="radio"/> No
	*是否能正确初始化 initialization returns OK	<input type="radio"/> Yes	<input type="radio"/> No