# Arduino Library For BC7215 Universal IR encoder/decoder

## Introduction

This driver library provides an interface between the Arduino system and the BC7215 universal infrared codec chip, enabling various operations targeting the BC7215 chip. To use this driver library, a basic understanding of the BC7215 chip is required; please refer to the BC7215 datasheet for more information.

This driver library can operate on any Arduino board that offers a hardware serial port, as well as on Arduino boards using a software serial port, but the software serial port must be set to a baud rate of 19200, 8 data bits, and **2 stop bits**.

## Installation

In the Arduino IDE, select *"Sketch --> Include Library --> Add .ZIP Library"*, and choose the *bc7215.zip* package file to complete the installation. After installation, you can see the example programs provided with the library under *File --> Examples* in the IDE.

## Data Types

The driver library defines the data types:

```
struct bc7215DataMaxPkt_t{
    word bitLen;
    byte data[BC7215_MAX_DATA_BYTE_LENGTH];
};
```

This type defines the maximum data packet that can be processed, and the maximum data packet length that can be handled is defined in the **library configuration file** (see the "Advanced Applications" section.)

```
struct bc7215FormatPkt_t{
    union {
        struct {
            byte sig  : 6;
            byte c56k : 1;
            byte noCA : 1;
        } bits;
        byte byte;
    } signature;
    byte format[32];
};
```

This type defines the format information packet of the BC7215.

## Usage

To use this driver library, simply add the following line at the very top of your sketch:

```
#include "bc7215.h"
```

The BC7215 uses a serial connection to Arduino and also requires 2 digital I/O pins for connecting the MOD and BUSY signals.

To use it, you must first create an instance of BC7215, like so:

```
BC7215 irModule(Serial1, 6, 7);
```

This line of code creates an instance of BC7215 named *irModule*. In the parameters, *Serial1* refers to the Serial port connected to the BC7215, "6" is the I/O pin connected to MOD, and "7" is the I/O pin connected to the BUSY signal.

The MOD and BUSY signals do not have to be connected to the Arduino in every applications. In some cases, you might only use the BC7215 for either receiving or transmitting, in which case the MOD signal can be directly connected to either VCC or GND. Similarly, if you are not using the infrared transmission function, the BUSY signal can be left unconnected.

For these situations, when creating an instance of BC7215, special parameters represent these specific hardware connections:

- `BC7215::MOD_HIGH`
- `BC7215::MOD_LOW`
- `BC7215::BUSY_NC`

These respectively represent the MOD connected to a **fixed high** level, MOD connected to a **fixed low** level, and BUSY **not connected**. Example:

```
BC7215 irModule(Serial1, BC7215::MOD_HIGH, BC7215::BUSY_NC);
```

In the initialization function `setup()`, you need to configure the serial port parameters connected to the BC7215:

```
Serial1.begin(19200, SERIAL_8N2);
```

The MOD and BUSY I/O pins are automatically initialized by the library.

# Interface Functions

The BC7215 class provides:

1. State Control Functions
2. Query Functions
3. Reception-related Functions
4. Transmission-related Functions
5. Utility Functions

## 1. State Control Functions

These functions allow users to control the working state of the BC7215.

```
void setRx();
```

Sets the BC7215's working mode to receive (infrared decoding) mode. The actual operation sets the MOD signal to high. If this function is called while the BC7215 is in the process of infrared transmission, the BC7215 may need to complete the transmission of the current infrared bits before switching to receive mode, which could take up to 20ms. Users should ensure that no other operations are performed on the BC7215 during this period.

```
void setTx();
```

Sets the BC7215's working mode to transmit (infrared encoding) mode. The actual operation sets the MOD signal to low. After calling this function, it may take up to 2ms for the BC7215 chip to complete the mode switch. Users should ensure that no other operations are performed on the BC7215 during this period.

```
void setShutDown();
```

In transmission mode, sets the BC7215 to shut down mode. After calling this function, the command F7 00 is sent to the BC7215. This function only takes effect in transmission mode. If called in receive mode, since the last data sent is 0x00, the BC7215's receive mode will switch to simple mode (for details, please refer to the BC7215 datasheet). After calling this function, users can query the command execution status with **cmdCompleted**().

```
void setRxMode(byte mode);
```

In receive mode, sets the receiving decoding mode (for details, please refer to the BC7215 datasheet). The lowest two bits of `mode` determine the working mode after the command is executed.

## 2.Query Functions

These functions allow users to query the working status of the BC7215.

```
bool dataReady();
```

Queries whether valid infrared data has been received from the BC7215. Returns 1 (true) if valid data is present, otherwise returns 0 (false).

This query is only valid in receive mode; if called in transmission mode, it will always return 0.

If the reception function is disabled in the *library configuration file*, this function is not available. There are three circumstances under which the status of valid infrared data will be cleared:

1. New data from BC7215 is received via the serial port.
2. The `getData()` or `getRaw()` functions are called.
3. The `clrData()` function is called to clear the data packet.

```
bool formatReady();
```

Queries whether format data has been received from the BC7215. Returns 1 (true) if valid format data is present, otherwise returns 0 (false). This query is only valid in receive mode; if called in transmission mode, it will always return 0.

In composite mode, BC7215 outputs format information of the received infrared signal in addition to the raw data. Generally, if format data is received, raw data has also been received previously, but there is a case where the total length of the raw data plus format information exceeds the length of the driver library's buffer. In this case, the previously received raw data will be overwritten, leaving only the format information usable.

To obtain the raw data in such cases, BC7215 needs to be set to simple mode to re-receive the infrared signal. If the reception function is disabled in the configuration, this function is not available. The status of valid format data will be cleared under three circumstances:

1. New data from  BC7215 is received via the serial port.
2. The `getFormat()` function is called.
3. The `clrFormat()` function is called to clear the format packet.

```
bool cmdCompleted();
```

In transmission mode, queries whether a command has been completed. A return value of 1 (true) indicates that the command has been completed; 0 (false) indicates it has not. In receive mode, this function will always return 1.

This can include transmission commands and shutdown commands. The BC7215 chip has an internal 16-byte reception buffer. Although the rate of infrared transmission is relatively low, for data amounts within 16 bytes, the transmission function will return almost immediately, but the actual data transmission process, completed by the BC7215 chip, takes longer. Sometimes users need to know when the transmission is complete, for example, to ensure a specific interval between two transmissions. This function can be used to query the completion time of the last transmission. The shutdown command executes immediately, and this function can query whether the BC7215 has entered the shutdown state.

## 3. Reception-related Functions

```
word getLen();
```

Gets the length of the received raw data packet in bits, which is, the `bitLen` in `bc7215DataMaxPkt_t`.

The actual length of data copied by the `getData()` function corresponds to the number of bytes for this value +2. For example, if `bitLen` is 9, the raw data requires 2 bytes, and the actual number of bytes copied will be 4.

If the raw data packet is unavailable, the result will be 0. This function allows users to check the length of data before retrieving raw data to prevent memory overflow or to dynamically allocate memory. If the reception function is disabled in the library configuration, this function is not available.

```
word dpktSize();
```

Gets the size of the received data packet in bytes.

This function is equivalent to the one that gets the bit length but returns the size of the entire data packet in bytes, saving users the efforts of conversion. Since the data packet length is determined by the infrared transmitter, it's possible to receive data exceeding expectations. Users can check the size of the data packet before retrieval to prevent memory overflow. If the reception function is disabled in the library configuration, this function is not available.

```
byte getData(bc7215DataMaxPkt_t& target);
```

Function for reading the data packet.

The input parameter is a variable of type `bc7215DataMaxPkt_t`. After executing the command, the received data packet will be copied into this variable. The return value is the status byte of the data packet, which can be used to quickly determine the status of the received data.

If the data in the cache is no longer available (e.g., it has been overwritten by new data), it will return 0xff.

This function clears the `dataReady()` status. Users must ensure that the target size can accommodate the received data; executing this function with insufficient memory space for the data can lead to memory overflow and unpredictable consequences. Users should first use `getLen()` or `dpktSize()` to get the data length, allocate memory or check space size, then call this function. If the reception function is disabled in the library configuration, this function is not available.

```
word getRaw(void* addr, word size);
```

Function for retrieving the raw data from the received data packet.

Similar to the above data packet reading function, but this function only reads out the raw payload data without the bit length information and can output to any address, making it more suitable for use in infrared communication. `size` is the number of bytes to read, which may not necessarily match the amount of data received; it can be less than or more than the actual number of bytes received. If `size` is less, it returns data up to the value of `size`; if `size` is more, it only reads out the actual number of bytes received. The return value is the actual number of bytes read. If the reception function is disabled in the library configuration, this function is not available.

```
byte getFormat(bc7215FormatPkt_t& target);
```

Function for reading the format data packet.

The input parameter is a variable of type `bc7215FormatPkt_t`. After executing the command, the received format data packet will be copied into this variable. The return value is the signature byte of the format data packet. If the data in the library cache is no longer available (e.g., it has been overwritten by new data), it will return 0xff. This function clears the `formatReady()` status. If the reception function is disabled in the library configuration, this function is not available.

## 4. Transmission-related Functions

```
void loadFormat(const bc7215FormatPkt_t& source);
```

Loads format data into the BC7215 chip.

After calling this function, the format information data pointed to by `source` will be loaded into the BC7215 chip. If the transmission function is disabled in the library configuration, this function is not available.

```
void irTx(const bc7215DataMaxPkt_t& source);
```

Transmits infrared data. Calling this function will cause the BC7215 to transmit the data packet `source` via infrared.

The format used for transmission is the last loaded format. If switching from receive to transmit mode without having loaded a format, the format of the last received infrared signal will be used. If the data to be transmitted is less than 16 bytes (128 bits), this function will return immediately after writing the data to the BC7215's internal buffer; if more than 16 bytes, it returns after the last 16 bytes are written to the buffer. The specific time required for infrared transmission depends on the infrared modulation format used, typically several ms per byte. The `cmdCompleted()` function can be used to query whether the infrared transmission is complete. If the transmission function is disabled in the configuration, this function is not available.

```
void sendRaw(const void* source, word size);
```

Sends raw data.

Similar to `irTx()`, but `source` can be any type of data, not limited to the raw data packet format, with `size` being the number of bytes to be sent. This function is suitable for data communication.

## 5. Utility Functions

The BC7215 class also provides some utility functions. These functions do not directly operate the BC7215 chip but are common operations that users frequently need when using BC7215.

```
void BC7215::setC56K(bc7215FormatPkt_t& target);
```

Sets the control bit of the format data packet feature byte. After setting, the C56K bit will be set, and after loading the data packet, BC7215 will use a 56K carrier for transmission.

```
void BC7215::clrC56K(bc7215FormatPkt_t& target);
```

Clears the control bit of the format data packet feature byte. After setting, the C56K bit will be cleared (reset to default state), and after loading the data packet, BC7215 will use a 38K carrier for transmission.

```
void BC7215::setNOCA(bc7215FormatPkt_t& target);
```

Sets the control bit of the format data packet feature byte. After setting, the NOCA bit will be set, and after loading the data packet, BC7215 output will not use a carrier, producing only high and low level outputs.

```
void BC7215::clrNOCA(bc7215FormatPkt_t& target);
```

Clears the control bit of the format data packet feature byte. After setting, the NOCA bit will be cleared (reset to default state), and after loading the data packet, BC7215 will use a 38k or 56k carrier for output.

```
byte BC7215::crc8(byte* data, word len);
```

CRC8 calculation function.

If BC7215 is used for data communication, it might be necessary to add CRC checks to data packets to improve reliability. Since data packets suitable for BC7215 are generally small, (recommended data length is within 16 bytes,) and because the infrared communication rate is low, using an 8-bit CRC check is more appropriate. It prevents errors without significantly increasing communication time.

The function has two input parameters: a pointer to the data and the length of the data to calculate the CRC for. The length here refers to byte length, not bit length, and the function calculates CRC by byte. The return value is the calculated CRC result.

The CRC calculation polynomial, default value is 0x07, defined in `bc7215_config.h`, can be modified by users as needed.

```
word BC7215::calSize(const bc7215DataMaxPkt_t& dataPkt);
```

Calculates the size of a data packet.

This function returns the size of the actual data packet in `dataPkt`, including both the raw data part and the bit length part, in bytes.

```
void BC7215::copyDpkt(void* target, bc7215DataMaxPkt_t& source);
```

Data packet copy function.

This function copies the data packet from `source` to the address of `target`. The copy operation only copies the size of the actual data packet in `source`. It is worth noting that this function supports the overlap of `source` and `target`, which effectively moves the data packet, for example, `target`'s address can be just 1 byte different from `source`, equivalent to moving the data packet forward or backward by one byte in memory.

```
bool compareDpkt(byte sig, const bc7215DataMaxPkt_t& pkt1, const
bc7215DataMaxPkt_t& pkt2);
```

Compares two data packets. This function compares whether the valid data in two packets are the same. If the same, the function returns `1` (true); if different, returns `0` (false).

Infrared data may not end in a complete byte, and this function supports the comparison of incomplete bytes. Due to different encoding formats, data may be MSB-first or LSB-first, so the signature byte is needed to determine the data alignment. The function assumes the signature bytes of both packets are the same. If the two packets have different modulation methods, their data lengths and data are most likely different.

# Advanced Applications

The driver library includes a configuration file `bc7215_config.h` located in the `libraries/bc7215` directory of the Arduino root directory. It defines some parameters related to this driver library. Advanced users can adjust these values according to their needs to better suit their projects. The main configuration parameters are as follows:

`ENABLE_RECEIVING`

This parameter controls whether to enable the driver library's receiving / decoding processing function, including receiving data packets, receiving format packets, etc. This parameter defaults to `1`, which means enabled. If users do not need BC7215's receiving decoding function, it can be modified to `0`. Disabling the receiving function also makes related functions unavailable. Since the receiving decoding function occupies most of the required memory and program content, disabling it will significantly reduce the size and memory footprint of this driver library.

`ENABLE_FORMAT`

Under the receiving function, there's a further control parameter for enabling the reception of format information. The default value is `1`, which means enabled. If not needed, it can be changed to `0`. Some applications, such as data communication, do not need to acquire format information of the infrared signal. In such cases, this function can be turned off, saving 33 bytes of RAM and some program space, as the format information packet is 33 bytes long.

`ENABLE_TRANSMITTING`

This parameter controls whether to enable the infrared transmission-related functions, defaulting to `1` for enabled status. Changing it to `0` can disable these functions, saving some program space.

`BC7215_MAX_RX_DATA_SIZE`

Represents the maximum length of  payload in a data packets that can be received, in bytes, ranging from 1 to 512.

In real world the actual length of data emitted by a remote controller, for audio-visual equipment, is generally within 8 bytes, and for air conditioners, generally within 32 bytes. The larger this defined value, the more memory the driver library will occupy.

`BC7215_CRC8_POLY`

The library provides a utility function for CRC-8 calculation. This parameter defines the polynomial value for CRC-8 calculation, with the default value being 0x07. Users can modify it according to their needs.