

Erriez Oregon THN128 433MHz temperature sensor library for Arduino  
1.1.0

Generated by Doxygen 1.8.17



<b>1 Main Page</b>	<b>1</b>
<b>2 Module Index</b>	<b>5</b>
2.1 Modules . . . . .	5
<b>3 Class Index</b>	<b>7</b>
3.1 Class List . . . . .	7
<b>4 File Index</b>	<b>9</b>
4.1 File List . . . . .	9
<b>5 Module Documentation</b>	<b>11</b>
5.1 data macro's . . . . .	11
5.1.1 Detailed Description . . . . .	11
5.1.2 Macro Definition Documentation . . . . .	11
5.1.2.1 GET_CHANNEL . . . . .	11
5.1.2.2 GET_CRC . . . . .	12
5.1.2.3 GET_ROL_ADDR . . . . .	12
5.1.2.4 GET_TEMP . . . . .	12
5.1.2.5 LOW_BAT_BIT . . . . .	12
5.1.2.6 SET_CHANNEL . . . . .	13
5.1.2.7 SET_CRC . . . . .	13
5.1.2.8 SET_ROL_ADDR . . . . .	13
5.1.2.9 SET_TEMP . . . . .	13
5.1.2.10 SIGN_BIT . . . . .	13
<b>6 Class Documentation</b>	<b>15</b>
6.1 OregonTHN128Data_t Struct Reference . . . . .	15
6.1.1 Detailed Description . . . . .	15
6.1.2 Member Data Documentation . . . . .	15
6.1.2.1 channel . . . . .	15
6.1.2.2 lowBattery . . . . .	16
6.1.2.3 rawData . . . . .	16
6.1.2.4 rollingAddress . . . . .	16
6.1.2.5 temperature . . . . .	16
<b>7 File Documentation</b>	<b>17</b>
7.1 src/ErriezOregonTHN128.c File Reference . . . . .	17
7.1.1 Detailed Description . . . . .	18
7.1.2 Function Documentation . . . . .	18
7.1.2.1 OregonTHN128_CheckCRC() . . . . .	18
7.1.2.2 OregonTHN128_DataToRaw() . . . . .	18
7.1.2.3 OregonTHN128_RawToData() . . . . .	19
7.1.2.4 OregonTHN128_TempToString() . . . . .	19

7.2 src/ErriezOregonTHN128Receive.c File Reference . . . . .	20
7.2.1 Detailed Description . . . . .	21
7.2.2 Enumeration Type Documentation . . . . .	21
7.2.2.1 RxState_t . . . . .	21
7.2.3 Function Documentation . . . . .	21
7.2.3.1 OregonTHN128_Available() . . . . .	21
7.2.3.2 OregonTHN128_Read() . . . . .	22
7.2.3.3 OregonTHN128_RxBegin() . . . . .	22
7.3 src/ErriezOregonTHN128Receive.h File Reference . . . . .	22
7.3.1 Detailed Description . . . . .	24
7.3.2 Function Documentation . . . . .	24
7.3.2.1 OregonTHN128_Available() . . . . .	24
7.3.2.2 OregonTHN128_Read() . . . . .	25
7.3.2.3 OregonTHN128_RxBegin() . . . . .	25
7.4 src/ErriezOregonTHN128Transmit.c File Reference . . . . .	25
7.4.1 Detailed Description . . . . .	26
7.4.2 Function Documentation . . . . .	26
7.4.2.1 OregonTHN128_Transmit() . . . . .	26
7.4.2.2 OregonTHN128_TxBegin() . . . . .	27
7.4.2.3 OregonTHN128_TxEnd() . . . . .	27
7.4.2.4 OregonTHN128_TxRawData() . . . . .	27
7.5 src/ErriezOregonTHN128Transmit.h File Reference . . . . .	28
7.5.1 Detailed Description . . . . .	29
7.5.2 Function Documentation . . . . .	29
7.5.2.1 OregonTHN128_Transmit() . . . . .	29
7.5.2.2 OregonTHN128_TxBegin() . . . . .	29
7.5.2.3 OregonTHN128_TxRawData() . . . . .	29
<b>Index</b>	<b>31</b>

# Chapter 1

## Main Page

### Oregon THN128 433MHz temperature transmit/receive library for Arduino

This is a 433MHz wireless 3-channel Oregon THN128 temperature transmit/receive Arduino library for ATmega328, ESP8266 and ESP32 emulating v1 protocol:

#### Transmit / receive hardware

This Arduino library is optimized for low-power ATmega328 microcontroller (AVR architectures like `Arduino UNO` and `Pro Mini 3.3V 8MHz` boards).

#### Temperature transmitter on the left breadboard:

- Pro-Mini 3V3 8MHz.
- Genuine DS18B20 temperature sensor.
- STX802 low-power 433MHz transmitter.

#### Receiver on the right breadboard:

- SRX882 low-power 433MHz receiver.
- SSD1306 I2C 128x64 OLED display.
- Pro-Mini 3V3 8MHz.

#### Supported microcontrollers

- ATmega328 AVR designed for low-power
- ESP8266
- ESP32
- Other microcontrollers are not tested and may or may not work

## Hardware notes

Supported hardware:

- AVR designed for low-power
- ESP8266
- ESP32
- For low-power transmitters, a `Pro Mini 3V3 8MHz` bare board with ATmega328 microcontroller is highly recommended. The board has no serial interface chip which reduces continuous power consumption. An external FTDI232 - USB serial interface should be connected for serial console / programming. (See red PCB on the picture) The SMD power LED should be desoldered from the Pro Mini to reduce continuous power consumption.
- A transmitter with (protected) 1500mA 18650 battery can operate for at least 6 months with `LowPower.h` functionality implemented. (By sending the temperature every 30 seconds)
- Changing the BOD (Brown Out Detection) fuse to 1.8V allows operation between 1.8 and 4.2V 18650 battery. (Explanation beyond the scope of this project)
- 1 to 3 temperature transmitters are supported, similar to the original Oregon THN128 temperature transmitters.
- Check [list of counterfeit DS18B20 chips](#), because this makes a huge difference in accuracy and read errors at 3.3V. Many DS18B20 chips from Aliexpress are counterfeit and won't work reliable at voltages below 3.3V.
- [NiceRF Wireless Technology Co., Ltd.](#) sells high quality 433MHz transmit (STX802) and receiver modules (STX882) with a good range.
- A 18650 battery (with protection circuit) should be connected directly to the VCC pin (not VIN).
- The voltage regulator can be desoldered from the pro-micro board when not used for more power reduction.

## Oregon Protocol

A packet is sent twice:

Data (see header file `ErriezOregonTHN128Receive.h`):

- Byte 0:
  - Bit 0..3: Rolling address (Random value after power cycle)
  - Bit 6..7: Channel: (0 = channel 1 .. 2 = channel 3)
- Byte 1:
  - Bit 0..3: TH3
  - Bit 4..7: TH2
- Byte 2:
  - Bit 0..3: TH1
  - Bit 5: Sign
  - Bit 7: Low battery
- Byte 3:
  - Bit 0..7: CRC

## Example low power receive

```
{c++}
#include <LowPower.h>
#include <ErriezOregonTHN128Receive.h>
// Connect RF receive to Arduino pin 2 (INT0) or pin 3 (INT1)
#define RF_RX_PIN    2
void printReceivedData(OregonTHN128Data_t *data)
{
    bool negativeTemperature = false;
    static uint32_t rxCount = 0;
    int16_t tempAbs;
    char msg[80];
    // Convert to absolute temperature
    tempAbs = data->temperature;
    if (tempAbs < 0) {
        negativeTemperature = true;
        tempAbs *= -1;
    }
    snprintf_P(msg, sizeof(msg),
        PSTR("RX %lu: Rol: %d, Channel %d, Temp: %s%d.%d, Low batt: %d (0x%08lx)"),
        rxCount++,
        data->rollingAddress, data->channel,
        (negativeTemperature ? "-" : ""), (tempAbs / 10), (tempAbs % 10), data->lowBattery,
        data->rawData);
    Serial.println(msg);
}
void setup()
{
    // Initialize serial port
    Serial.begin(115200);
    Serial.println(F("Oregon THN128 433MHz temperature receive"));
    // Turn LED on
    pinMode(LED_BUILTIN, OUTPUT);
    digitalWrite(LED_BUILTIN, HIGH);
    // Initialize receiver
    OregonTHN128_RxBegin(RF_RX_PIN);
}
void loop()
{
    OregonTHN128Data_t data;
    // Check temperature received
    if (OregonTHN128_Available()) {
        digitalWrite(LED_BUILTIN, LOW);

        // Read temperature
        OregonTHN128_Read(&data);
        // Print received data
        printReceivedData(&data);
        // Wait ~30 seconds before receiving next temperature
        Serial.flush();
        LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
        LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
        LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
        LowPower.powerDown(SLEEP_2S, ADC_OFF, BOD_OFF);
        digitalWrite(LED_BUILTIN, HIGH);
        // Enable receive
        OregonTHN128_RxEnable();
    }
}
```

## Example low power transmit

```
{c++}
#include <LowPower.h>
#include <ErriezOregonTHN128Transmit.h>
// Pin defines (Any DIGITAL pin)
#define RF_TX_PIN    3
OregonTHN128Data_t data = {
    .rawData = 0,           // Raw data filled in by library
    .rollingAddress = 5,    // Rolling address 0..7
    .channel = 1,           // Channel 1, 2 or 3
    .temperature = 0,        // Temperature -99.9 .. 99.9 multiplied by 10
    .lowBattery = false,    // Low battery true or false
};
#ifdef __cplusplus
extern "C" {
#endif
// Function is called from library
void delay100ms()
{
    // Blink LED within 100ms space between two packets
    digitalWrite(LED_BUILTIN, HIGH);
    LowPower.powerDown(SLEEP_15MS, ADC_OFF, BOD_OFF);
}
```

```

    digitalWrite(LED_BUILTIN, LOW);
    LowPower.powerDown(SLEEP_60MS, ADC_OFF, BOD_OFF);
    LowPower.powerDown(SLEEP_15MS, ADC_OFF, BOD_OFF);
}
#ifdef __cplusplus
}
#endif
void setup()
{
    // Initialize built-in LED
    pinMode(LED_BUILTIN, OUTPUT);
    digitalWrite(LED_BUILTIN, LOW);
    // Initialize pins
    OregonTHN128_TxBegin(RF_TX_PIN);
}
void loop()
{
    // Set temperature
    data.temperature = 123; //12.3'C
    // Send temperature
    OregonTHN128_Transmit(&data);
    // Wait ~30 seconds before sending next temperature
    LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
    LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
    LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
    LowPower.powerDown(SLEEP_4S, ADC_OFF, BOD_OFF);
}

```

## Library Changes

### v1.1.0

The callback function `void delay100ms()` has been removed as this was not compatible with ESP32. The application should change the code to:

```

{c++}
    // Send temperature twice with 100ms delay between packets
    OregonTHN128_Transmit(&data);
    delay(100);
    OregonTHN128_Transmit(&data);

```

AVR targets can replace `delay(100)` with LowPower usage:

```

{c++}
    LowPower.powerDown(SLEEP_15MS, ADC_OFF, BOD_OFF);
    LowPower.powerDown(SLEEP_60MS, ADC_OFF, BOD_OFF);
    LowPower.powerDown(SLEEP_15MS, ADC_OFF, BOD_OFF);

```

## Saleae Logic Analyzer

`capture` from the Oregon THN128 can be opened with <https://www.saleae.com/downloads/>.

## Generated Arduino Library Doxygen Documentation

- [Online Doxygen HTML](#)
- [Doxygen PDF](#)

## MIT License

This project is published under [MIT license](#) with an additional end user agreement (next section).

## End User Agreement :ukraine:

End users shall accept the [End User Agreement](#) holding export restrictions to Russia to stop the WAR before using this project.



## Chapter 2

# Module Index

### 2.1 Modules

Here is a list of all modules:

data macro's . . . . .	11
------------------------	----



## Chapter 3

# Class Index

### 3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">OregonTHN128Data_t</a>	
Data structure . . . . .	15



## Chapter 4

# File Index

### 4.1 File List

Here is a list of all documented files with brief descriptions:

src/ <a href="#">ErriezOregonTHN128.c</a>	
Oregon THN128 433MHz temperature transmit/receive library for Arduino . . . . .	<a href="#">17</a>
src/ <b>ErriezOregonTHN128.h</b> . . . . .	<b>??</b>
src/ <a href="#">ErriezOregonTHN128Receive.c</a>	
Oregon THN128 433MHz temperature transmit/receive library for Arduino . . . . .	<a href="#">20</a>
src/ <a href="#">ErriezOregonTHN128Receive.h</a>	
Oregon THN128 433MHz temperature receive library for Arduino . . . . .	<a href="#">22</a>
src/ <a href="#">ErriezOregonTHN128Transmit.c</a>	
Oregon THN128 433MHz temperature transmit library for Arduino . . . . .	<a href="#">25</a>
src/ <a href="#">ErriezOregonTHN128Transmit.h</a>	
Oregon THN128 433MHz temperature transmit library for Arduino . . . . .	<a href="#">28</a>



## Chapter 5

# Module Documentation

### 5.1 data macro's

#### Macros

- #define SET\_ROL\_ADDR(x) (((x) & 0x07) << 0)
- #define GET\_ROL\_ADDR(x) (((x) & 0x07) << 0)
- #define SET\_CHANNEL(x) (((x) - 1) & 0x03) << 6)
- #define GET\_CHANNEL(x) (((x) >> 6) & 0x03) + 1)
- #define SET\_TEMP(x)
- #define GET\_TEMP(x)
- #define SIGN\_BIT (1UL << 21)
- #define LOW\_BAT\_BIT (1UL << 23)
- #define SET\_CRC(x) ((uint32\_t)(x) << 24)
- #define GET\_CRC(x) ((x) >> 24)

#### 5.1.1 Detailed Description

#### 5.1.2 Macro Definition Documentation

##### 5.1.2.1 GET\_CHANNEL

```
#define GET_CHANNEL(  
    x ) (((x) >> 6) & 0x03) + 1)
```

Get channel

Definition at line 49 of file ErriezOregonTHN128.c.

### 5.1.2.2 GET\_CRC

```
#define GET_CRC(  
    x ) ((x) >> 24)
```

Get CRC

Definition at line 69 of file ErriezOregonTHN128.c.

### 5.1.2.3 GET\_ROL\_ADDR

```
#define GET_ROL_ADDR(  
    x ) (((x) & 0x07) << 0)
```

Get rolling address

Definition at line 44 of file ErriezOregonTHN128.c.

### 5.1.2.4 GET\_TEMP

```
#define GET_TEMP(  
    x )
```

**Value:**

```
(((((x) >> 16) & 0x0f) * 100) + \  
(((x) >> 12) & 0x0f) * 10) + \  
(((x) >> 8) & 0x0f))
```

Get temperature

Definition at line 56 of file ErriezOregonTHN128.c.

### 5.1.2.5 LOW\_BAT\_BIT

```
#define LOW_BAT_BIT (1UL << 23)
```

Low battery bit

Definition at line 64 of file ErriezOregonTHN128.c.



### 5.1.2.6 SET\_CHANNEL

```
#define SET_CHANNEL(  
    x ) (((x) - 1) & 0x03) << 6)
```

Set channel

Definition at line 47 of file ErriezOregonTHN128.c.

### 5.1.2.7 SET\_CRC

```
#define SET_CRC(  
    x ) ((uint32_t)(x) << 24)
```

Set CRC

Definition at line 67 of file ErriezOregonTHN128.c.

### 5.1.2.8 SET\_ROL\_ADDR

```
#define SET_ROL_ADDR(  
    x ) ((x) & 0x07) << 0)
```

Set rolling address

Definition at line 42 of file ErriezOregonTHN128.c.

### 5.1.2.9 SET\_TEMP

```
#define SET_TEMP(  
    x )
```

**Value:**

```
(((((uint32_t)(x) / 100) % 10)) << 16) | \  
(((uint32_t)(x) / 10) % 10) << 12) | \  
((x) % 10) << 8)
```

Set temperature

Definition at line 52 of file ErriezOregonTHN128.c.

### 5.1.2.10 SIGN\_BIT

```
#define SIGN_BIT (1UL << 21)
```

Sign bit

Definition at line 61 of file ErriezOregonTHN128.c.



## Chapter 6

# Class Documentation

### 6.1 OregonTHN128Data\_t Struct Reference

Data structure.

```
#include <ErriezOregonTHN128.h>
```

#### Public Attributes

- uint32\_t [rawData](#)
- uint8\_t [rollingAddress](#)
- uint8\_t [channel](#)
- int16\_t [temperature](#)
- bool [lowBattery](#)

#### 6.1.1 Detailed Description

Data structure.

Definition at line 63 of file ErriezOregonTHN128.h.

#### 6.1.2 Member Data Documentation

##### 6.1.2.1 channel

```
uint8_t OregonTHN128Data_t::channel
```

Channel

Definition at line 66 of file ErriezOregonTHN128.h.

#### 6.1.2.2 lowBattery

```
bool OregonTHN128Data_t::lowBattery
```

Low battery indication

Definition at line 68 of file ErriezOregonTHN128.h.

#### 6.1.2.3 rawData

```
uint32_t OregonTHN128Data_t::rawData
```

Raw data

Definition at line 64 of file ErriezOregonTHN128.h.

#### 6.1.2.4 rollingAddress

```
uint8_t OregonTHN128Data_t::rollingAddress
```

Rolling address

Definition at line 65 of file ErriezOregonTHN128.h.

#### 6.1.2.5 temperature

```
int16_t OregonTHN128Data_t::temperature
```

Temperature

Definition at line 67 of file ErriezOregonTHN128.h.

The documentation for this struct was generated from the following file:

- src/ErriezOregonTHN128.h

## Chapter 7

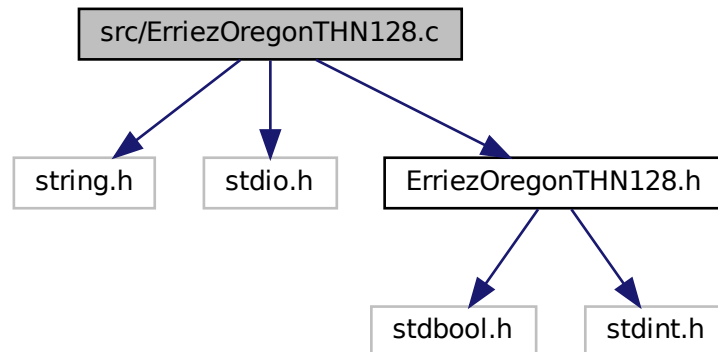
# File Documentation

### 7.1 src/ErriezOregonTHN128.c File Reference

Oregon THN128 433MHz temperature transmit/receive library for Arduino.

```
#include <string.h>
#include <stdio.h>
#include "ErriezOregonTHN128.h"
```

Include dependency graph for ErriezOregonTHN128.c:



### Macros

- `#define SET_ROL_ADDR(x) (((x) & 0x07) << 0)`
- `#define GET_ROL_ADDR(x) (((x) & 0x07) << 0)`
- `#define SET_CHANNEL(x) (((x) - 1) & 0x03) << 6)`
- `#define GET_CHANNEL(x) (((x) >> 6) & 0x03) + 1)`
- `#define SET_TEMP(x)`
- `#define GET_TEMP(x)`
- `#define SIGN_BIT (1UL << 21)`
- `#define LOW_BAT_BIT (1UL << 23)`
- `#define SET_CRC(x) ((uint32_t)(x) << 24)`
- `#define GET_CRC(x) ((x) >> 24)`

## Functions

- bool [OregonTHN128\\_CheckCRC](#) (uint32\_t rawData)  
*Verify checksum.*
- void [OregonTHN128\\_TempToString](#) (char \*temperatureStr, uint8\_t temperatureStrLen, int16\_t temperature)  
*Convert temperature to string.*
- uint32\_t [OregonTHN128\\_DataToRaw](#) ([OregonTHN128Data\\_t](#) \*data)  
*Convert data structure to 32-bit raw data.*
- bool [OregonTHN128\\_RawToData](#) (uint32\_t rawData, [OregonTHN128Data\\_t](#) \*data)  
*Convert 32-bit raw data to [OregonTHN128Data\\_t](#) structure.*

### 7.1.1 Detailed Description

Oregon THN128 433MHz temperature transmit/receive library for Arduino.

Source: <https://github.com/Erriez/ErriezOregonTHN128> Documentation: <https://erriez.github.io/ErriezOregonTHN128>

### 7.1.2 Function Documentation

#### 7.1.2.1 OregonTHN128\_CheckCRC()

```
bool OregonTHN128_CheckCRC (
    uint32_t rawData )
```

Verify checksum.

##### Parameters

<i>rawData</i>	32-bit raw data input
----------------	-----------------------

##### Returns

true: Success, false: error

Definition at line 101 of file ErriezOregonTHN128.c.

#### 7.1.2.2 OregonTHN128\_DataToRaw()

```
uint32_t OregonTHN128_DataToRaw (
    OregonTHN128Data_t * data )
```

Convert data structure to 32-bit raw data.

## Parameters

<i>data</i>	Input
-------------	-------

## Returns

Output

Definition at line 141 of file ErriezOregonTHN128.c.

### 7.1.2.3 OregonTHN128\_RawToData()

```
bool OregonTHN128_RawToData (
    uint32_t rawData,
    OregonTHN128Data_t * data )
```

Convert 32-bit raw data to [OregonTHN128Data\\_t](#) structure.

## Parameters

<i>rawData</i>	32-bit input
<i>data</i>	output

## Returns

CRC true: Success, false: error

Definition at line 180 of file ErriezOregonTHN128.c.

### 7.1.2.4 OregonTHN128\_TempToString()

```
void OregonTHN128_TempToString (
    char * temperatureStr,
    uint8_t temperatureStrLen,
    int16_t temperature )
```

Convert temperature to string.

## Parameters

<i>temperatureStr</i>	Character buffer
<i>temperatureStrLen</i>	Size of character buffer
<i>temperature</i>	Input temperature

Definition at line 118 of file ErriezOregonTHN128.c.

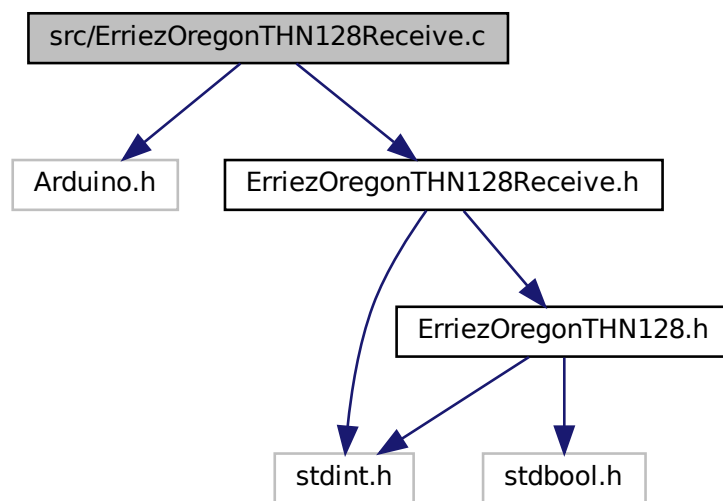
## 7.2 src/ErriezOregonTHN128Receive.c File Reference

Oregon THN128 433MHz temperature transmit/receive library for Arduino.

```
#include <Arduino.h>
```

```
#include "ErriezOregonTHN128Receive.h"
```

Include dependency graph for ErriezOregonTHN128Receive.c:



### Enumerations

- enum `RxState_t` {  
`StateSearchSync` = 0, `StateMid0` = 1, `StateMid1` = 2, `StateEnd` = 3,  
`StateRxComplete` = 4 }

*Receive state.*

### Functions

- void `rfPinChange` (void)  
*RF pin level change.*
- void `OregonTHN128_RxBegin` (uint8\_t extIntPin)  
*Initialize receiver pin.*
- void `OregonTHN128_RxEnable` ()  
*Receive enable.*
- void `OregonTHN128_RxDisable` ()  
*Receive disable.*
- bool `OregonTHN128_Available` ()  
*Check if data received.*
- bool `OregonTHN128_Read` (OregonTHN128Data\_t \*data)  
*Read data.*



## 7.2.1 Detailed Description

Oregon THN128 433MHz temperature transmit/receive library for Arduino.

Source: <https://github.com/Erriez/ErriezOregonTHN128> Documentation: <https://erriez.github.io/ErriezOregonTHN128>

## 7.2.2 Enumeration Type Documentation

### 7.2.2.1 RxState\_t

enum `RxState_t`

Receive state.

Enumerator

StateSearchSync	Search for sync
StateMid0	Sample at the middle of a pulse part 1
StateMid1	Sample at the middle of a pulse part 2
StateEnd	Sample at the end of a pulse to store bit
StateRxComplete	Receive complete

Definition at line 44 of file `ErriezOregonTHN128Receive.c`.

## 7.2.3 Function Documentation

### 7.2.3.1 OregonTHN128\_Available()

`bool OregonTHN128_Available ( )`

Check if data received.

Return values

<i>true</i>	Data received
<i>false</i>	No data available

Definition at line 358 of file `ErriezOregonTHN128Receive.c`.

### 7.2.3.2 OregonTHN128\_Read()

```
bool OregonTHN128_Read (
    OregonTHN128Data_t * data )
```

Read data.

#### Parameters

<i>data</i>	Structure <a href="#">OregonTHN128Data_t</a> output
-------------	---

#### Return values

<i>true</i>	Data received
<i>false</i>	No data available

Definition at line 373 of file `ErriezOregonTHN128Receive.c`.

### 7.2.3.3 OregonTHN128\_RxBegin()

```
void OregonTHN128_RxBegin (
    uint8_t extIntPin )
```

Initialize receiver pin.

Connect RX pin to an external interrupt pin such as INT0 (D2) or INT1 (D3)

#### Parameters

<i>extIntPin</i>	
------------------	--

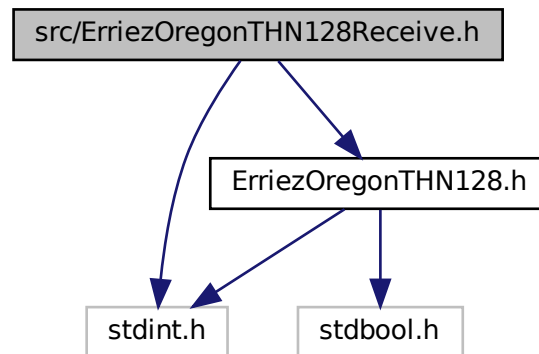
Definition at line 324 of file `ErriezOregonTHN128Receive.c`.

## 7.3 src/ErriezOregonTHN128Receive.h File Reference

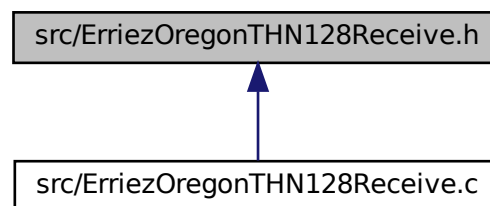
Oregon THN128 433MHz temperature receive library for Arduino.

```
#include <stdint.h>
#include "ErriezOregonTHN128.h"
```

Include dependency graph for ErriezOregonTHN128Receive.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void `OregonTHN128_RxBegin` (uint8\_t extIntPin)  
*Initialize receiver pin.*
- void `OregonTHN128_RxEnable` ()  
*Receive enable.*
- void `OregonTHN128_RxDisable` ()  
*Receive disable.*
- bool `OregonTHN128_Available` (void)  
*Check if data received.*
- bool `OregonTHN128_Read` (OregonTHN128Data\_t \*data)  
*Read data.*

### 7.3.1 Detailed Description

Oregon THN128 433MHz temperature receive library for Arduino.

Source: <https://github.com/Erriez/ErriezOregonTHN128> Documentation: <https://erriez.github.io/ErriezOregonTHN128>

Protocol:

Transmit temperature twice every 30 seconds:

Bit: 0 7 0 7 0 7 0 7 +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ | PREA | SYNC | Byte 0 | Byte 1 | Byte 2 | Byte 3 |  
| PREA | SYNC | Byte 0 | ... +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ | <----- 144ms -----  
-> | <- 100ms -> | 30 sec

Logic '0': Logic '1': +---+ +---+ | | +---+ +---+ 1400 1500 1500 1400 (us)

PREA: Preamble 12x logic '1', 3000us low

SYNC: +-----+ | |

- +-----+ 5500us 5500us

Byte 0:

- Bit 0..3: Rolling address (Random value after power cycle)
- Bit 6..7: Channel: (0 = channel 1 .. 2 = channel 3)

Byte 1:

- Bit 0..3: TH3
- Bit 4..7: TH2

Byte 2:

- Bit 0..3: TH1
- Bit 5: Sign
- Bit 7: Low battery

Byte 3:

- Bit 0..7: CRC

Example: Rolling address = 5, channel = 1, temperature = 27.8 `C, low battery = false TH1 = 2, TH2 = 7, TH3 = 8:  
Byte 0: 0x05 Byte 1: 0x78 Byte 2: 0x02 Byte 3: 0x7f

Bits in time: PRE=1 S B0=0x05 B1=0x78 B2=0x02 B3=0x7f 111111111111 S 10100000 00011110 01000000 11111110

### 7.3.2 Function Documentation

#### 7.3.2.1 OregonTHN128\_Available()

```
bool OregonTHN128_Available ( )
```

Check if data received.

## Return values

<i>true</i>	Data received
<i>false</i>	No data available

Definition at line 358 of file ErriezOregonTHN128Receive.c.

### 7.3.2.2 OregonTHN128\_Read()

```
bool OregonTHN128_Read (
    OregonTHN128Data_t * data )
```

Read data.

## Parameters

<i>data</i>	Structure <a href="#">OregonTHN128Data_t</a> output
-------------	---

## Return values

<i>true</i>	Data received
<i>false</i>	No data available

Definition at line 373 of file ErriezOregonTHN128Receive.c.

### 7.3.2.3 OregonTHN128\_RxBegin()

```
void OregonTHN128_RxBegin (
    uint8_t extIntPin )
```

Initialize receiver pin.

Connect RX pin to an external interrupt pin such as INT0 (D2) or INT1 (D3)

## Parameters

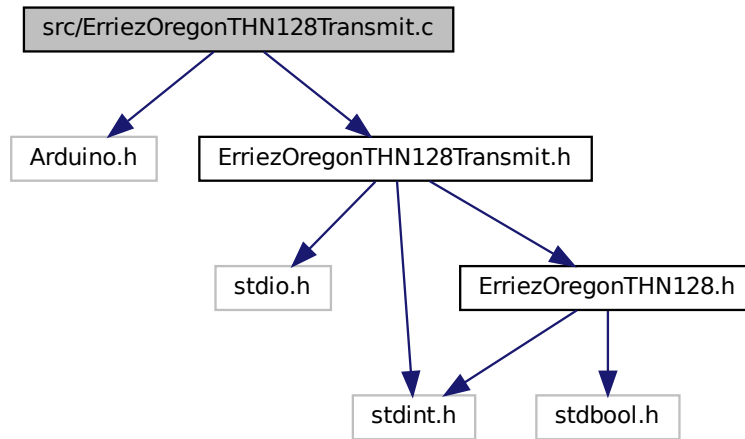
<i>extIntPin</i>	
------------------	--

Definition at line 324 of file ErriezOregonTHN128Receive.c.

## 7.4 src/ErriezOregonTHN128Transmit.c File Reference

Oregon THN128 433MHz temperature transmit library for Arduino.

```
#include <Arduino.h>
#include "ErriezOregonTHN128Transmit.h"
Include dependency graph for ErriezOregonTHN128Transmit.c:
```



## Functions

- void `OregonTHN128_TxBegin` (uint8\_t rfTxPin)  
*Transmit begin.*
- void `OregonTHN128_TxEnd` (void)  
*Disable transmit.*
- void `OregonTHN128_TxRawData` (uint32\_t rawData)  
*Transmit data.*
- void `OregonTHN128_Transmit` (`OregonTHN128Data_t` \*data)  
*Transmit Transmit data.*

### 7.4.1 Detailed Description

Oregon THN128 433MHz temperature transmit library for Arduino.

Source: <https://github.com/Erriez/ErriezOregonTHN128> Documentation: <https://erriez.github.io/ErriezOregonTHN128>

### 7.4.2 Function Documentation

#### 7.4.2.1 `OregonTHN128_Transmit()`

```
void OregonTHN128_Transmit (
    OregonTHN128Data_t * data )
```

Transmit Transmit data.

The application should call `OregonTHN128_TxRawData()` twice at 100ms interval.

**Parameters**

<i>data</i>	Oregon THN128 input structure
-------------	-------------------------------

Definition at line 292 of file ErriezOregonTHN128Transmit.c.

**7.4.2.2 OregonTHN128\_TxBegin()**

```
void OregonTHN128_TxBegin (  
    uint8_t rfTxPin )
```

Transmit begin.

Connect rfTxPin to any DIGITAL pin

**Parameters**

<i>rfTxPin</i>	Arduino transmit pin
----------------	----------------------

Definition at line 248 of file ErriezOregonTHN128Transmit.c.

**7.4.2.3 OregonTHN128\_TxEnd()**

```
void OregonTHN128_TxEnd (  
    void )
```

Disable transmit.

Set transmit pin to input

Definition at line 259 of file ErriezOregonTHN128Transmit.c.

**7.4.2.4 OregonTHN128\_TxRawData()**

```
void OregonTHN128_TxRawData (  
    uint32_t rawData )
```

Transmit data.

**Parameters**

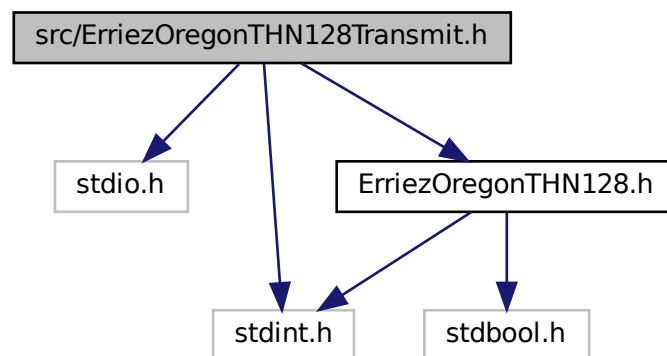
<i>rawData</i>	32-bit raw data input
----------------	-----------------------

Definition at line 270 of file ErriezOregonTHN128Transmit.c.

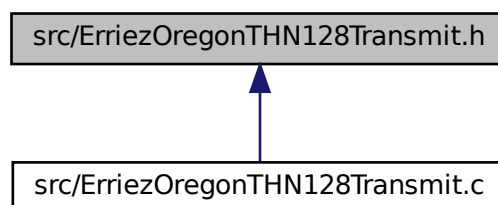
## 7.5 src/ErriezOregonTHN128Transmit.h File Reference

Oregon THN128 433MHz temperature transmit library for Arduino.

```
#include <stdio.h>
#include <stdint.h>
#include "ErriezOregonTHN128.h"
Include dependency graph for ErriezOregonTHN128Transmit.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- void `OregonTHN128_TxBegin` (uint8\_t rTxPin)  
*Transmit begin.*
- void `OregonTHN128_TxRawData` (uint32\_t rawData)  
*Transmit data.*
- void `OregonTHN128_Transmit` (OregonTHN128Data\_t \*data)  
*Transmit Transmit data.*



## 7.5.1 Detailed Description

Oregon THN128 433MHz temperature transmit library for Arduino.

Source: <https://github.com/Erriez/ErriezOregonTHN128> Documentation: <https://erriez.github.io/ErriezOregonTHN128>

## 7.5.2 Function Documentation

### 7.5.2.1 OregonTHN128\_Transmit()

```
void OregonTHN128_Transmit (
    OregonTHN128Data_t * data )
```

Transmit Transmit data.

The application should call [OregonTHN128\\_TxRawData\(\)](#) twice at 100ms interval.

#### Parameters

<i>data</i>	Oregon THN128 input structure
-------------	-------------------------------

Definition at line 292 of file ErriezOregonTHN128Transmit.c.

### 7.5.2.2 OregonTHN128\_TxBegin()

```
void OregonTHN128_TxBegin (
    uint8_t rfTxPin )
```

Transmit begin.

Connect rfTxPin to any DIGITAL pin

#### Parameters

<i>rfTxPin</i>	Arduino transmit pin
----------------	----------------------

Definition at line 248 of file ErriezOregonTHN128Transmit.c.

### 7.5.2.3 OregonTHN128\_TxRawData()

```
void OregonTHN128_TxRawData (
    uint32_t rawData )
```

Transmit data.

#### Parameters

<i>rawData</i>	32-bit raw data input
----------------	-----------------------

Definition at line 270 of file ErriezOregonTHN128Transmit.c.

# Index

- channel
  - [OregonTHN128Data\\_t, 15](#)
- data macro's, [11](#)
  - [GET\\_CHANNEL, 11](#)
  - [GET\\_CRC, 11](#)
  - [GET\\_ROL\\_ADDR, 12](#)
  - [GET\\_TEMP, 12](#)
  - [LOW\\_BAT\\_BIT, 12](#)
  - [SET\\_CHANNEL, 12](#)
  - [SET\\_CRC, 13](#)
  - [SET\\_ROL\\_ADDR, 13](#)
  - [SET\\_TEMP, 13](#)
  - [SIGN\\_BIT, 13](#)
- ErriezOregonTHN128.c
  - [OregonTHN128\\_CheckCRC, 18](#)
  - [OregonTHN128\\_DataToRaw, 18](#)
  - [OregonTHN128\\_RawToData, 19](#)
  - [OregonTHN128\\_TempToString, 19](#)
- ErriezOregonTHN128Receive.c
  - [OregonTHN128\\_Available, 21](#)
  - [OregonTHN128\\_Read, 21](#)
  - [OregonTHN128\\_RxBegin, 22](#)
  - [RxState\\_t, 21](#)
  - [StateEnd, 21](#)
  - [StateMid0, 21](#)
  - [StateMid1, 21](#)
  - [StateRxComplete, 21](#)
  - [StateSearchSync, 21](#)
- ErriezOregonTHN128Receive.h
  - [OregonTHN128\\_Available, 24](#)
  - [OregonTHN128\\_Read, 25](#)
  - [OregonTHN128\\_RxBegin, 25](#)
- ErriezOregonTHN128Transmit.c
  - [OregonTHN128\\_Transmit, 26](#)
  - [OregonTHN128\\_TxBegin, 27](#)
  - [OregonTHN128\\_TxEnd, 27](#)
  - [OregonTHN128\\_TxRawData, 27](#)
- ErriezOregonTHN128Transmit.h
  - [OregonTHN128\\_Transmit, 29](#)
  - [OregonTHN128\\_TxBegin, 29](#)
  - [OregonTHN128\\_TxRawData, 29](#)
- GET\_CHANNEL
  - data macro's, [11](#)
- GET\_CRC
  - data macro's, [11](#)
- GET\_ROL\_ADDR
  - data macro's, [12](#)
- GET\_TEMP
  - data macro's, [12](#)
- LOW\_BAT\_BIT
  - data macro's, [12](#)
- lowBattery
  - [OregonTHN128Data\\_t, 15](#)
- OregonTHN128\_Available
  - [ErriezOregonTHN128Receive.c, 21](#)
  - [ErriezOregonTHN128Receive.h, 24](#)
- OregonTHN128\_CheckCRC
  - [ErriezOregonTHN128.c, 18](#)
- OregonTHN128\_DataToRaw
  - [ErriezOregonTHN128.c, 18](#)
- OregonTHN128\_RawToData
  - [ErriezOregonTHN128.c, 19](#)
- OregonTHN128\_Read
  - [ErriezOregonTHN128Receive.c, 21](#)
  - [ErriezOregonTHN128Receive.h, 25](#)
- OregonTHN128\_RxBegin
  - [ErriezOregonTHN128Receive.c, 22](#)
  - [ErriezOregonTHN128Receive.h, 25](#)
- OregonTHN128\_TempToString
  - [ErriezOregonTHN128.c, 19](#)
- OregonTHN128\_Transmit
  - [ErriezOregonTHN128Transmit.c, 26](#)
  - [ErriezOregonTHN128Transmit.h, 29](#)
- OregonTHN128\_TxBegin
  - [ErriezOregonTHN128Transmit.c, 27](#)
  - [ErriezOregonTHN128Transmit.h, 29](#)
- OregonTHN128\_TxEnd
  - [ErriezOregonTHN128Transmit.c, 27](#)
- OregonTHN128\_TxRawData
  - [ErriezOregonTHN128Transmit.c, 27](#)
  - [ErriezOregonTHN128Transmit.h, 29](#)
- OregonTHN128Data\_t, [15](#)
  - channel, [15](#)
  - lowBattery, [15](#)
  - rawData, [16](#)
  - rollingAddress, [16](#)
  - temperature, [16](#)
- rawData
  - [OregonTHN128Data\\_t, 16](#)
- rollingAddress
  - [OregonTHN128Data\\_t, 16](#)
- RxState\_t
  - [ErriezOregonTHN128Receive.c, 21](#)
- SET\_CHANNEL

- data macro's, [12](#)
- SET\_CRC
  - data macro's, [13](#)
- SET\_ROL\_ADDR
  - data macro's, [13](#)
- SET\_TEMP
  - data macro's, [13](#)
- SIGN\_BIT
  - data macro's, [13](#)
- src/ErriezOregonTHN128.c, [17](#)
- src/ErriezOregonTHN128Receive.c, [20](#)
- src/ErriezOregonTHN128Receive.h, [22](#)
- src/ErriezOregonTHN128Transmit.c, [25](#)
- src/ErriezOregonTHN128Transmit.h, [28](#)
- StateEnd
  - ErriezOregonTHN128Receive.c, [21](#)
- StateMid0
  - ErriezOregonTHN128Receive.c, [21](#)
- StateMid1
  - ErriezOregonTHN128Receive.c, [21](#)
- StateRxComplete
  - ErriezOregonTHN128Receive.c, [21](#)
- StateSearchSync
  - ErriezOregonTHN128Receive.c, [21](#)
- temperature
  - OregonTHN128Data\_t, [16](#)