Erriez Oregon THN128 433MHz temperature sensor library for Arduino

1.1.0

Generated by Doxygen 1.9.8

Chapter 1

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 using the (reverse-engineered) Oregon THN128 v1 protocol:

Transmit / receive hardware

This Arduino library can be used with 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

Arduino Examples

- Oregon THN128 Receive
- Oregon THN128 Receive SSD1306 OLED
- Oregon THN128 Transmit random temperature
- Oregon THN128 Transmit DS1820 1-wire temperature sensor
- Oregon THN128 ESP32 MQTT Homeassistant

ESP32 with MQTT and Homeassistant

The Erriez_Oregon_THN128_ESP32_MQTT_Homeassistant.ino sketch can be used with Homeassistant integration.

Example screenshot Homeassistant dasboard:

Follow the steps below:

1. Configure Homeassistant MQTT in configuration.yaml:

```
mqtt:
  discovery_prefix: ha
  # Enable when using SSL:
  # certificate: /ssl/ca.crt
  # client_cert: /ssl/client.crt
  # client_key: /ssl/client.key
```

1. MQTT broker hostname, username and password should be configured in Homeassistant | Settings | Devices | MQTT.
2. Configure the listed macro's in the example, build and run from the Arduino IDE. The following Oregon THN128 entities are automatically registered after a succesful MQTT connection:

- sensor.oregon_thn128_ch1
- sensor.oregon_thn128_ch2
- sensor.oregon_thn128_ch3
- sensor.oregon_thn128_battery

1. Configure Homeassistant dashboard configuration file:

- Homeassistant Dashboard YAML

Hardware Design 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

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:

```
++
   // 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:

```
++
   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

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

Chapter 5

Topic 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

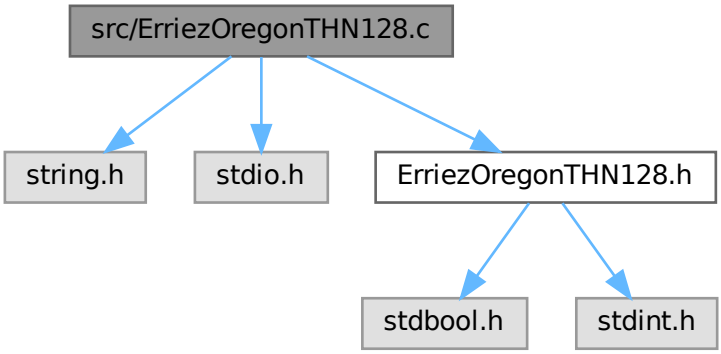
File Documentation

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)

  Cnonvert 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

Definition in file ErriezOregonTHN128.c.

7.1.2 Function Documentation

7.1.2.1 OregonTHN128_CheckCRC()

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

Verify checksum.

Parameters

rawData	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

data	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 )
```

Cnonvert 32-bit raw data to OregonTHN128Data_t structure.

Parameters

rawData	32-bit input
data	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

temperatureStr	Character buffer
temperatureStrLen	Size of character buffer
temperature	Input temperature

Definition at line 118 of file ErriezOregonTHN128.c.

7.2 ErriezOregonTHN128.c

Go to the documentation of this file.

```
00001 /*
00002  * MIT License
00003  *
00004  * Copyright (c) 2020-2026 Erriez
00005  *
00006  * Permission is hereby granted, free of charge, to any person obtaining a copy
00007  * of this software and associated documentation files (the "Software"), to deal
00008  * in the Software without restriction, including without limitation the rights
00009  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010  * copies of the Software, and to permit persons to whom the Software is
00011  * furnished to do so, subject to the following conditions:
00012  *
00013  * The above copyright notice and this permission notice shall be included in all
00014  * copies or substantial portions of the Software.
00015  *
00016  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022  * SOFTWARE.
00023  */
00024
00033 #include <string.h>
00034 #include <stdio.h>
00035 #include "ErriezOregonTHN128.h"
00036
00042 #define SET_ROL_ADDR(x)      (((x) & 0x07) << 0)
00044 #define GET_ROL_ADDR(x)      (((x) & 0x07) << 0)
00045
00047 #define SET_CHANNEL(x)       ((((x) - 1) & 0x03) << 6)
00049 #define GET_CHANNEL(x)       ((((x) >> 6) & 0x03) + 1)
00050
00052 #define SET_TEMP(x)          ((((((uint32_t)(x) / 100) % 10)) << 16) | \
00053                               ((((uint32_t)(x) / 10) % 10) << 12) | \
00054                               (((x) % 10) << 8))
00056 #define GET_TEMP(x)          (((((x) >> 16) & 0x0f) * 100) + \
00057                               ((((x) >> 12) & 0x0f) * 10) + \
00058                               (((x) >> 8) & 0x0f))
00059
00061 #define SIGN_BIT             (1UL << 21)
00062
00064 #define LOW_BAT_BIT          (1UL << 23)
00065
00067 #define SET_CRC(x)           ((uint32_t)(x) << 24)
00069 #define GET_CRC(x)           ((x) >> 24)
00070
00081 static uint8_t calcCrc(uint32_t rawData)
00082 {
00083     uint16_t crc;
00084
00085     /* Add Bytes 0, 1 and 2 */
00086     crc = ((rawData >> 16) & 0xff) + ((rawData >> 8) & 0xff) + ((rawData >> 0) & 0xff);
00087     /* Add most significant Byte of the CRC to the final CRC */
00088     crc = (crc >> 8) + (crc & 0xff);
00089
00090     /* Return 8-bit CRC */
00091     return (uint8_t)crc;
00092 }
00093
00101 bool OregonTHN128_CheckCRC(uint32_t rawData)
00102 {
00103     return calcCrc(rawData) == GET_CRC(rawData);
00104 }
00105
00106 /*---------------------------------------------------------------------------------------------*/
00107 /*                                    Public functions                                         */
00108 /*---------------------------------------------------------------------------------------------*/
00118 void OregonTHN128_TempToString(char *temperatureStr, uint8_t temperatureStrLen, int16_t temperature)
00119 {
00120     bool tempNegative = false;
00121     int tempAbs;
00122
00123     /* Convert temperature without using float to string */
00124     tempAbs = temperature;
00125     if (temperature < 0) {
00126         tempNegative = true;
00127         tempAbs *= -1;
00128     }
00129
00130     snprintf(temperatureStr, temperatureStrLen, "%s%d.%d",
```

```
00131             tempNegative ? "-" : "", (tempAbs / 10), tempAbs % 10);
00132 }
00133
00141 uint32_t OregonTHN128_DataToRaw(OregonTHN128Data_t *data)
00142 {
00143     uint32_t rawData;
00144
00145     /* Rolling address 0..7 */
00146     rawData = SET_ROL_ADDR(data->rollingAddress);
00147
00148     /* Set channel 1..3 */
00149     rawData |= SET_CHANNEL(data->channel);
00150
00151     /* Set temperature -999..999 */
00152     if (data->temperature < 0) {
00153         rawData |= SIGN_BIT;
00154         rawData |= SET_TEMP(data->temperature * -1);
00155     } else {
00156         rawData |= SET_TEMP(data->temperature);
00157     }
00158
00159     /* Low battery bit */
00160     if (data->lowBattery) {
00161         rawData |= LOW_BAT_BIT;
00162     }
00163
00164     /* Calculate CRC */
00165     rawData |= SET_CRC(calcCrc(rawData));
00166
00167     /* Return 32-bit raw data */
00168     return rawData;
00169 }
00170
00180 bool OregonTHN128_RawToData(uint32_t rawData, OregonTHN128Data_t *data)
00181 {
00182     memset(data, 0, sizeof(OregonTHN128Data_t));
00183
00184     /* Set data structure */
00185     data->rawData = rawData;
00186     data->rollingAddress = GET_ROL_ADDR(rawData);
00187     data->channel = GET_CHANNEL(rawData);
00188     data->temperature = GET_TEMP(rawData);
00189     if (rawData & SIGN_BIT) {
00190         data->temperature *= -1;
00191     }
00192     data->lowBattery = (rawData & LOW_BAT_BIT) ? true : false;
00193
00194     /* Return CRC success or failure */
00195     return calcCrc(rawData) == GET_CRC(rawData);
00196 }
```

7.3 ErriezOregonTHN128.h

```
00001 /*
00002  * MIT License
00003  *
00004  * Copyright (c) 2020-2026 Erriez
00005  *
00006  * Permission is hereby granted, free of charge, to any person obtaining a copy
00007  * of this software and associated documentation files (the "Software"), to deal
00008  * in the Software without restriction, including without limitation the rights
00009  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010  * copies of the Software, and to permit persons to whom the Software is
00011  * furnished to do so, subject to the following conditions:
00012  *
00013  * The above copyright notice and this permission notice shall be included in all
00014  * copies or substantial portions of the Software.
00015  *
00016  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022  * SOFTWARE.
00023  */
00024
00025 #ifndef ERRIEZ_OREGON_THN128_H_
00026 #define ERRIEZ_OREGON_THN128_H_
00027
00028 /* Check platform */
00029 #if !defined(ARDUINO_ARCH_AVR) && !defined(ARDUINO_ARCH_ESP8266) && !defined(ARDUINO_ARCH_ESP32)
00030 #error "Platform not supported."
```

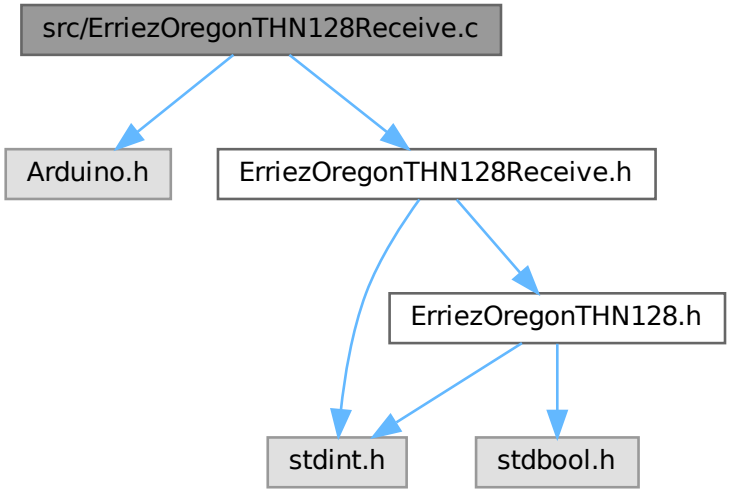
```
00031 #endif
00032
00033 #ifdef __cplusplus
00034 extern "C" {
00035 #endif
00036
00037 #include <stdbool.h>
00038 #include <stdint.h>
00039
00040 /* Timing micro's in micro seconds */
00041 #define T_RX_TOLERANCE_US       400
00042 #define T_PREAMBLE_SPACE_US     3000
00043 #define T_SYNC_US               5500
00044 #define T_BIT_US                1450
00045 #define T_SPACE_FRAMES_MS       100
00046
00047 #define T_SYNC_H_MIN        (T_SYNC_US - T_RX_TOLERANCE_US)
00048 #define T_SYNC_H_MAX        (T_SYNC_US + T_RX_TOLERANCE_US)
00049
00050 #define T_SYNC_L_MIN_0      (T_SYNC_US + T_BIT_US - T_RX_TOLERANCE_US)
00051 #define T_SYNC_L_MAX_0      (T_SYNC_US + T_BIT_US + T_RX_TOLERANCE_US)
00052 #define T_SYNC_L_MIN_1      (T_SYNC_US - T_RX_TOLERANCE_US)
00053 #define T_SYNC_L_MAX_1      (T_SYNC_US + T_RX_TOLERANCE_US)
00054
00055 #define T_BIT_SHORT_MIN     (T_BIT_US - T_RX_TOLERANCE_US)
00056 #define T_BIT_SHORT_MAX     (T_BIT_US + T_RX_TOLERANCE_US)
00057 #define T_BIT_LONG_MIN      ((T_BIT_US * 2) - T_RX_TOLERANCE_US)
00058 #define T_BIT_LONG_MAX      ((T_BIT_US * 2) + T_RX_TOLERANCE_US)
00059
00063 typedef struct {
00064     uint32_t rawData;
00065     uint8_t rollingAddress;
00066     uint8_t channel;
00067     int16_t temperature;
00068     bool lowBattery;
00069 } OregonTHN128Data_t;
00070
00071 /* Public functions */
00072 bool OregonTHN128_CheckCRC(uint32_t rawData);
00073 void OregonTHN128_TempToString(char *temperatureStr, uint8_t temperatureStrLen, int16_t temperature);
00074 uint32_t OregonTHN128_DataToRaw(OregonTHN128Data_t *data);
00075 bool OregonTHN128_RawToData(uint32_t rawData, OregonTHN128Data_t *data);
00076
00077 #ifdef __cplusplus
00078 }
00079 #endif
00080
00081 #endif /* ERRIEZ_OREGON_THN128_H_ */
```

7.4 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.

7.4.1 Detailed Description

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

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

Definition in file ErriezOregonTHN128Receive.c.

7.4.2 Enumeration Type Documentation

7.4.2.1 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.

7.4.3 Function Documentation

7.4.3.1 OregonTHN128_Available()

```
bool OregonTHN128_Available (
            void  )
```

Check if data received.

Return values

true	Data received
false	No data available

Definition at line 358 of file ErriezOregonTHN128Receive.c.

7.4.3.2 OregonTHN128_Read()

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

Read data.

Parameters

data	Structure OregonTHN128Data_t output

Return values

true	Data received

Return values

false	No data available

Definition at line 373 of file ErriezOregonTHN128Receive.c.

7.4.3.3 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

extIntPin	

Definition at line 324 of file ErriezOregonTHN128Receive.c.

7.4.3.4 OregonTHN128_RxDisable()

```
void OregonTHN128_RxDisable ( )
```

Receive disable.

Definition at line 345 of file ErriezOregonTHN128Receive.c.

7.4.3.5 OregonTHN128_RxEnable()

```
void OregonTHN128_RxEnable ( )
```

Receive enable.

Definition at line 336 of file ErriezOregonTHN128Receive.c.

7.4.3.6 rfPinChange()

```
void IRAM_ATTR rfPinChange (
            void  )
```

RF pin level change.

Definition at line 265 of file ErriezOregonTHN128Receive.c.

7.5 ErriezOregonTHN128Receive.c

Go to the documentation of this file.

```
00001 /*
00002  * MIT License
00003  *
00004  * Copyright (c) 2020-2026 Erriez
00005  *
00006  * Permission is hereby granted, free of charge, to any person obtaining a copy
00007  * of this software and associated documentation files (the "Software"), to deal
00008  * in the Software without restriction, including without limitation the rights
00009  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010  * copies of the Software, and to permit persons to whom the Software is
00011  * furnished to do so, subject to the following conditions:
00012  *
00013  * The above copyright notice and this permission notice shall be included in all
00014  * copies or substantial portions of the Software.
00015  *
00016  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022  * SOFTWARE.
00023  */
00024
00033 #include <Arduino.h>
00034
00035 #if defined(ARDUINO_ARCH_AVR)
00036 #include <avr/interrupt.h>
00037 #endif
00038
00039 #include "ErriezOregonTHN128Receive.h"
00040
00044 typedef enum {
00045     StateSearchSync = 0,
00046     StateMid0 = 1,
00047     StateMid1 = 2,
00048     StateEnd = 3,
00049     StateRxComplete = 4
00050 } RxState_t;
00051
00052 /* Static variables */
00053 static uint8_t _rxPin;
00054 static uint32_t _tPulseBegin;
00055 static uint16_t _tPinHigh;
00056 static uint16_t _tPinLow;
00057 static int8_t _rxBit;
00058 static volatile uint32_t _rxData;
00059 static volatile RxState_t _rxState = StateSearchSync;
00060
00061 /* Pin functions */
00062 #if defined(ARDUINO_ARCH_AVR)
00063 static uint8_t _rxPinPort;
00064 static uint8_t _rxPinBit;
00065
00072 #define RF_RX_PIN_INIT(rfRxPin) {                   \
00073     /* Save interrupt number of the RF pin */       \
00074     _rxPin = digitalPinToInterrupt(extIntPin);      \
00075     /* Save pin port and bit */                     \
00076     _rxPinPort = digitalPinToPort(extIntPin);       \
00077     _rxPinBit = digitalPinToBitMask(extIntPin);     \
00078 }
00079
00085 #define RF_RX_PIN_READ() (*portInputRegister(_rxPinPort) & _rxPinBit)
00086
00087 #elif defined(ARDUINO_ARCH_ESP8266) || defined(ARDUINO_ARCH_ESP32)
00088
00095 #define RF_RX_PIN_INIT(rfRxPin) { _rxPin = rfRxPin; }
00096
00102 #define RF_RX_PIN_READ() (digitalRead(_rxPin) ? HIGH : LOW)
00103
00104 #else
00105 #error "May work, but not tested on this target"
00106 #endif
00107
00108 /* Forward declaration */
00109 void rfPinChange(void);
00110
00111
00115 static void rxEnable()
00116 {
00117     /* Enable INTx change interrupt */
00118     attachInterrupt(_rxPin, rfPinChange, CHANGE);
```

```
00119
00120     /* Initialize with search for sync state */
00121     _rxState = StateSearchSync;
00122 }
00123
00127 static void rxDisable()
00128 {
00129     /* Disable INTx change interrupt */
00130     detachInterrupt(_rxPin);
00131 }
00132
00146 static bool isPulseInRange(uint16_t tPulse, uint16_t tMin, uint16_t tMax)
00147 {
00148     /* Check is pulse length between min and max time */
00149     if ((tPulse >= tMin) && (tPulse <= tMax)) {
00150         return true;
00151     } else {
00152         return false;
00153     }
00154 }
00155
00163 static bool findSync()
00164 {
00165     /* Read sync pulse */
00166     if (isPulseInRange(_tPinHigh, T_SYNC_H_MIN, T_SYNC_H_MAX)) {
00167         if (isPulseInRange(_tPinLow, T_SYNC_L_MIN_0, T_SYNC_L_MAX_0)) {
00168             _rxData = 0;
00169             _rxState = StateMid1;
00170             _rxBit = 1;
00171             return true;
00172         } else if (isPulseInRange(_tPinLow, T_SYNC_L_MIN_1, T_SYNC_L_MAX_1)) {
00173             _rxData = 0;
00174             _rxState = StateEnd;
00175             _rxBit = 0;
00176             return true;
00177         }
00178     }
00179
00180     return false;
00181 }
00182
00189 static void storeBit(bool one)
00190 {
00191     /* Store received bit */
00192     if (one) {
00193         _rxData |= (1UL << _rxBit);
00194     }
00195
00196     /* Check if all 32 data bits are received */
00197     _rxBit++;
00198     if (_rxBit >= 32) {
00199         if (OregonTHN128_CheckCRC(_rxData)) {
00200             _rxState = StateRxComplete;
00201             /* Disable receive */
00202             rxDisable();
00203         } else {
00204             _rxState = StateSearchSync;
00205         }
00206     }
00207 }
00208
00212 static void handlePulse()
00213 {
00214     if (isPulseInRange(_tPinHigh, T_BIT_SHORT_MIN, T_BIT_SHORT_MAX)) {
00215         if (_rxState == StateEnd) {
00216             _rxState = StateMid0;
00217             storeBit(1);
00218         } else if (_rxState == StateMid1) {
00219             _rxState = StateEnd;
00220         } else {
00221             _rxState = StateSearchSync;
00222         }
00223     } else if (isPulseInRange(_tPinHigh, T_BIT_LONG_MIN, T_BIT_LONG_MAX)) {
00224         if (_rxState == StateMid1) {
00225             _rxState = StateMid0;
00226             storeBit(1);
00227         } else {
00228             _rxState = StateSearchSync;
00229         }
00230     } else {
00231         _rxState = StateSearchSync;
00232     }
00233 }
00234
00238 static void handleSpace()
00239 {
00240     /* State machine */
```

```
00241     if (isPulseInRange(_tPinLow, T_BIT_SHORT_MIN, T_BIT_SHORT_MAX)) {
00242         if (_rxState == StateEnd) {
00243             _rxState = StateMid1;
00244             storeBit(0);
00245         } else if (_rxState == StateMid0) {
00246             _rxState = StateEnd;
00247         } else {
00248             _rxState = StateSearchSync;
00249         }
00250     } else if (isPulseInRange(_tPinLow, T_BIT_LONG_MIN, T_BIT_LONG_MAX)) {
00251         if (_rxState == StateMid0) {
00252             _rxState = StateMid1;
00253             storeBit(0);
00254         } else {
00255             _rxState = StateSearchSync;
00256         }
00257     } else {
00258         _rxState = StateSearchSync;
00259     }
00260 }
00261
00265 void IRAM_ATTR rfPinChange(void)
00266 {
00267     uint32_t tNow;
00268     uint16_t _tPulseLength;
00269     uint8_t rfPinHigh;
00270
00271     /* Return when previous completed receive is not read */
00272     if (_rxState == StateRxComplete) {
00273         return;
00274     }
00275
00276     /* Read absolute pulse time in us for sync */
00277     tNow = micros();
00278     if (tNow > _tPulseBegin) {
00279         _tPulseLength = tNow - _tPulseBegin;
00280     } else {
00281         _tPulseLength = _tPulseBegin - tNow;
00282     }
00283
00284     /* Ignore short pulses */
00285     if (_tPulseLength < T_RX_TOLERANCE_US) {
00286         return;
00287     }
00288     _tPulseBegin = tNow;
00289
00290     /* Get RF pin state */
00291     rfPinHigh = RF_RX_PIN_READ();
00292
00293     /* Store pulse (high) or space (low) length */
00294     if (rfPinHigh) {
00295         _tPinLow = _tPulseLength;
00296     } else {
00297         _tPinHigh = _tPulseLength;
00298     }
00299
00300     /* Always search for sync */
00301     if (findSync()) {
00302         return;
00303     }
00304
00305     /* Handle received pulse */
00306     if (_rxState != StateSearchSync) {
00307         if (rfPinHigh) {
00308             handleSpace();
00309         } else {
00310             handlePulse();
00311         }
00312     }
00313 }
00314
00315 /*-------------------------------------------------------------------------------------------------*/
00316 /*                                      Public functions                                           */
00317 /*-------------------------------------------------------------------------------------------------*/
00324 void OregonTHN128_RxBegin(uint8_t extIntPin)
00325 {
00326     /* Initialize RF RX pin */
00327     RF_RX_PIN_INIT(extIntPin);
00328
00329     /* Enable receive */
00330     rxEnable();
00331 }
00332
00336 void OregonTHN128_RxEnable()
00337 {
00338     /* Enable receive */
00339     rxEnable();
```

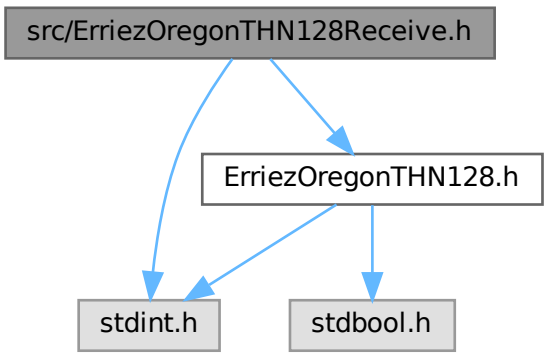
```
00340 }
00341
00345 void OregonTHN128_RxDisable()
00346 {
00347     /* Disable receive */
00348     rxDisable();
00349 }
00350
00358 bool OregonTHN128_Available()
00359 {
00360     /* Return receive complete */
00361     return (_rxState == StateRxComplete) ? true : false;
00362 }
00363
00373 bool OregonTHN128_Read(OregonTHN128Data_t *data)
00374 {
00375     if (OregonTHN128_Available()) {
00376         /* Convert raw 32-bit data to data structure */
00377         OregonTHN128_RawToData(_rxData, data);
00378         return true;
00379     } else {
00380         return false;
00381     }
00382 }
```

7.6 src/ErriezOregonTHN128Receive.h File Reference

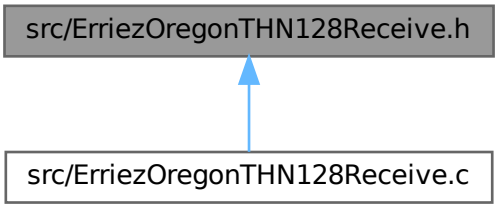
Oregon THN128 433MHz temperature receive library for Arduino.

```
#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.

7.6.1 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

Definition in file ErriezOregonTHN128Receive.h.

7.6.2 Macro Definition Documentation

7.6.2.1 IRAM_ATTR

```
#define IRAM_ATTR
```

Definition at line 94 of file ErriezOregonTHN128Receive.h.

7.6.3 Function Documentation

7.6.3.1 OregonTHN128_Available()

```
bool OregonTHN128_Available (
            void  )
```

Check if data received.

Return values

true	Data received
false	No data available

Definition at line 358 of file ErriezOregonTHN128Receive.c.

7.6.3.2 OregonTHN128_Read()

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

Read data.

Parameters

data	Structure OregonTHN128Data_t output

Return values

true	Data received
false	No data available

Definition at line 373 of file ErriezOregonTHN128Receive.c.

7.6.3.3 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

extIntPin	

Definition at line 324 of file ErriezOregonTHN128Receive.c.

7.6.3.4 OregonTHN128_RxDisable()

```
void OregonTHN128_RxDisable ( )
```

Receive disable.

Definition at line 345 of file ErriezOregonTHN128Receive.c.

7.6.3.5 OregonTHN128_RxEnable()

```
void OregonTHN128_RxEnable ( )
```

Receive enable.

Definition at line 336 of file ErriezOregonTHN128Receive.c.

7.7 ErriezOregonTHN128Receive.h

Go to the documentation of this file.

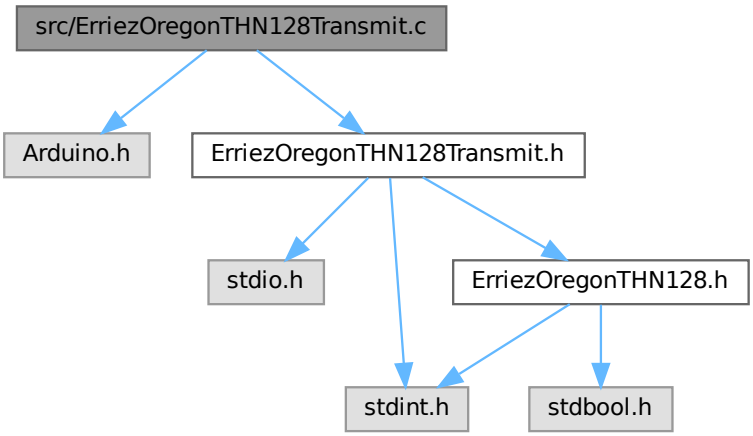
```
00001 /*
00002  * MIT License
00003  *
00004  * Copyright (c) 2020-2026 Erriez
00005  *
00006  * Permission is hereby granted, free of charge, to any person obtaining a copy
00007  * of this software and associated documentation files (the "Software"), to deal
00008  * in the Software without restriction, including without limitation the rights
00009  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010  * copies of the Software, and to permit persons to whom the Software is
00011  * furnished to do so, subject to the following conditions:
00012  *
00013  * The above copyright notice and this permission notice shall be included in all
00014  * copies or substantial portions of the Software.
00015  *
00016  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022  * SOFTWARE.
00023  */
00024
00086 #ifndef ERRIEZ_OREGON_THN128_RECEIVER_H_
00087 #define ERRIEZ_OREGON_THN128_RECEIVER_H_
00088
00089 #include <stdint.h>
00090 #include "ErriezOregonTHN128.h"
00091
00092 // Macro IRAM_ATTR is defined for ESP pin interrupts
00093 #ifndef IRAM_ATTR
00094 #define IRAM_ATTR
00095 #endif
00096
00097 #ifdef __cplusplus
00098 extern "C" {
00099 #endif
00100
00101 /* Public functions */
00102 void OregonTHN128_RxBegin(uint8_t extIntPin);
00103 void OregonTHN128_RxEnable();
00104 void OregonTHN128_RxDisable();
00105 bool OregonTHN128_Available(void);
00106 bool OregonTHN128_Read(OregonTHN128Data_t *data);
00107
00108 #ifdef __cplusplus
00109 }
00110 #endif
00111
00112 #endif /* ERRIEZ_OREGON_THN128_RECEIVER_H_ */
```

7.8 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.

7.8.1 Detailed Description

Oregon THN128 433MHz temperature transmit library for Arduino.

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

Definition in file ErriezOregonTHN128Transmit.c.

7.8.2 Function Documentation

7.8.2.1 OregonTHN128_Transmit()

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

Transmit Transmit data.

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

Parameters

data	Oregon THN128 input structure

Definition at line 292 of file ErriezOregonTHN128Transmit.c.

7.8.2.2 OregonTHN128_TxBegin()

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

Transmit begin.

Connect rfTxPin to any DIGITAL pin

Parameters

rfTxPin	Arduino transmit pin

Definition at line 248 of file ErriezOregonTHN128Transmit.c.

7.8.2.3 OregonTHN128_TxEnd()

```
void OregonTHN128_TxEnd (
            void  )
```

Disable transmit.

Set transmit pin to input

Definition at line 259 of file ErriezOregonTHN128Transmit.c.

7.8.2.4 OregonTHN128_TxRawData()

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

Transmit data.

Parameters

rawData	32-bit raw data input

Definition at line 270 of file ErriezOregonTHN128Transmit.c.

7.9 ErriezOregonTHN128Transmit.c

Go to the documentation of this file.

```
00001 /*
00002  * MIT License
00003  *
00004  * Copyright (c) 2020-2026 Erriez
00005  *
00006  * Permission is hereby granted, free of charge, to any person obtaining a copy
00007  * of this software and associated documentation files (the "Software"), to deal
00008  * in the Software without restriction, including without limitation the rights
00009  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010  * copies of the Software, and to permit persons to whom the Software is
00011  * furnished to do so, subject to the following conditions:
00012  *
00013  * The above copyright notice and this permission notice shall be included in all
00014  * copies or substantial portions of the Software.
00015  *
00016  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022  * SOFTWARE.
00023  */
00024
00033 #include <Arduino.h>
00034 #include "ErriezOregonTHN128Transmit.h"
00035
00036 /* Function prototypes */
00037 void delay100ms(void) __attribute__((weak));
00038 extern void delay100ms(void);
00039
00040 /* Static variables */
00041 #if defined(ARDUINO_ARCH_AVR)
00042 #include <util/delay.h>
00043
00044 static int8_t _rfTxPort = -1;
00045 static int8_t _rfTxBit = -1;
00046
00060 #define RF_TX_PIN_INIT(rfTxPin)         {             \
00061     _rfTxPort = digitalPinToPort(rfTxPin);            \
00062     _rfTxBit = digitalPinToBitMask(rfTxPin);          \
00063     *portModeRegister(_rfTxPort) |= _rfTxBit;         \
00064 }
00065
00070 #define RF_TX_PIN_DISABLE()               {           \
00071     if ((_rfTxPort >= 0) && (_rfTxBit >= 0)) {        \
00072         *portModeRegister(_rfTxPort) &= ~_rfTxBit;    \
00073     }                                                 \
00074 }
00075
00081 #define IS_RF_TX_PIN_INITIALIZED()  ((_rfTxPort >= 0) && (_rfTxBit >= 0))
00082
00087 #define RF_TX_PIN_HIGH()            { *portOutputRegister(_rfTxPort) |= _rfTxBit; }
00088
00093 #define RF_TX_PIN_LOW()             { *portOutputRegister(_rfTxPort) &= ~_rfTxBit; }
00094
00099 #define RF_TX_DELAY_US(us)          _delay_us(us)
00100
00105 #define RF_TX_DELAY_MS(ms)          _delay_ms(ms)
00106
00107 #elif defined(ARDUINO_ARCH_ESP8266) || defined(ARDUINO_ARCH_ESP32)
00108 static int8_t _rfTxPin = -1;
00109
00116 #define RF_TX_PIN_INIT(rfTxPin)  {                    \
00117     _rfTxPin= rfTxPin;                                \
00118     pinMode(_rfTxPin, OUTPUT);                        \
00119 }
00120
00125 #define RF_TX_PIN_DISABLE() {                         \
00126     if (_rfTxPin >= 0) {                              \
00127         pinMode(_rfTxPin, INPUT);                     \
00128     }                                                 \
00129 }
00130
00136 #define IS_RF_TX_PIN_INITIALIZED()  (_rfTxPin >= 0)
00137
00142 #define RF_TX_PIN_HIGH()            { digitalWrite(_rfTxPin, HIGH); }
00143
00148 #define RF_TX_PIN_LOW()             { digitalWrite(_rfTxPin, LOW); }
00149
00154 #define RF_TX_DELAY_US(us)          delayMicroseconds(us)
00155
```

```
00160 #define RF_TX_DELAY_MS(ms)          delay(ms)
00161 #else
00162 #error "May work, but not tested on this target"
00163 #endif
00164
00171 static void txSync()
00172 {
00173     /* Transmit sync pulse */
00174     RF_TX_PIN_HIGH();
00175     RF_TX_DELAY_US(T_SYNC_US);
00176     RF_TX_PIN_LOW();
00177     RF_TX_DELAY_US(T_SYNC_US);
00178 }
00179
00183 static void txBit0()
00184 {
00185     /* Transmit data bit 0 pulse */
00186     RF_TX_PIN_LOW();
00187     RF_TX_DELAY_US(T_BIT_US);
00188     RF_TX_PIN_HIGH();
00189     RF_TX_DELAY_US(T_BIT_US);
00190 }
00191
00195 static void txBit1()
00196 {
00197     /* Transmit data bit 1 pulse */
00198     RF_TX_PIN_HIGH();
00199     RF_TX_DELAY_US(T_BIT_US);
00200     RF_TX_PIN_LOW();
00201     RF_TX_DELAY_US(T_BIT_US);
00202 }
00203
00207 static void txDisable()
00208 {
00209     /* Transmit pin low */
00210     RF_TX_PIN_LOW();
00211 }
00212
00216 static void txPreamble()
00217 {
00218     /* Transmit 12 preamble bits 1 */
00219     for (uint8_t i = 0; i < 12; i++) {
00220         txBit1();
00221     }
00222     RF_TX_DELAY_US(T_PREAMBLE_SPACE_US);
00223 }
00224
00225 /* Transmit 32-bit data */
00226 static void txData(uint32_t data)
00227 {
00228     /* Transmit 32 data bits */
00229     for (uint8_t i = 0; i < 32; i++) {
00230         if (data & (1UL << i)) {
00231             txBit1();
00232         } else {
00233             txBit0();
00234         }
00235     }
00236 }
00237
00238 /*---------------------------------------------------------------------------------------------*/
00239 /*                                    Public functions                                         */
00240 /*---------------------------------------------------------------------------------------------*/
00248 void OregonTHN128_TxBegin(uint8_t rfTxPin)
00249 {
00250     /* Set RF transmit pin output */
00251     RF_TX_PIN_INIT(rfTxPin);
00252 }
00253
00259 void OregonTHN128_TxEnd(void)
00260 {
00261     /* Set RF transmit pin input */
00262     RF_TX_PIN_DISABLE();
00263 }
00264
00270 void OregonTHN128_TxRawData(uint32_t rawData)
00271 {
00272     /* Check RF transmit pin initialized */
00273     if (!IS_RF_TX_PIN_INITIALIZED()) {
00274         return;
00275     }
00276
00277     /* Transmit */
00278     txPreamble();
00279     txSync();
00280     txData(rawData);
00281     txDisable();
```

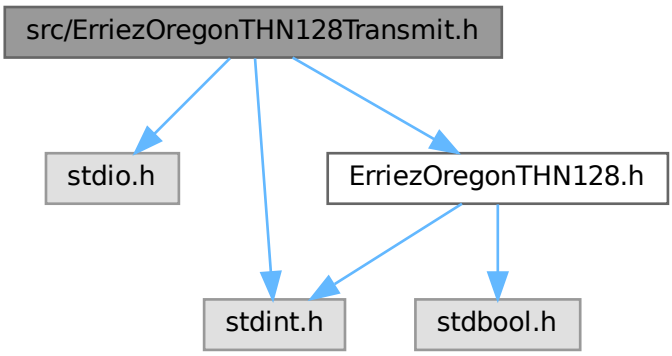
```
00282 }
00283
00292 void OregonTHN128_Transmit(OregonTHN128Data_t *data)
00293 {
00294     // Convert data structure to 32-bit raw data;
00295     data->rawData = OregonTHN128_DataToRaw(data);
00296
00297     // Send raw data
00298     OregonTHN128_TxRawData(data->rawData);
00299 }
```

7.10 src/ErriezOregonTHN128Transmit.h File Reference

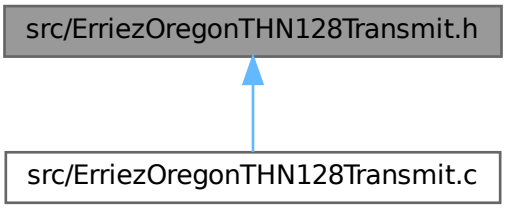
Oregon THN128 433MHz temperature transmit library for Arduino.

```
#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 rfTxPin)

  Transmit begin.
- void OregonTHN128_TxRawData (uint32_t rawData)

  Transmit data.
- void OregonTHN128_Transmit (OregonTHN128Data_t *data)

  Transmit Transmit data.

7.10.1 Detailed Description

Oregon THN128 433MHz temperature transmit library for Arduino.

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

Definition in file ErriezOregonTHN128Transmit.h.

7.10.2 Function Documentation

7.10.2.1 OregonTHN128_Transmit()

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

Transmit Transmit data.

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

Parameters

data	Oregon THN128 input structure

Definition at line 292 of file ErriezOregonTHN128Transmit.c.

7.10.2.2 OregonTHN128_TxBegin()

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

Transmit begin.

Connect rfTxPin to any DIGITAL pin

Parameters

rfTxPin	Arduino transmit pin

Definition at line 248 of file ErriezOregonTHN128Transmit.c.

7.10.2.3 OregonTHN128_TxRawData()

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

Transmit data.

Parameters

rawData	32-bit raw data input

Definition at line 270 of file ErriezOregonTHN128Transmit.c.

7.11 ErriezOregonTHN128Transmit.h

Go to the documentation of this file.

```
00001 /*
00002  * MIT License
00003  *
00004  * Copyright (c) 2020-2026 Erriez
00005  *
00006  * Permission is hereby granted, free of charge, to any person obtaining a copy
00007  * of this software and associated documentation files (the "Software"), to deal
00008  * in the Software without restriction, including without limitation the rights
00009  * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010  * copies of the Software, and to permit persons to whom the Software is
00011  * furnished to do so, subject to the following conditions:
00012  *
00013  * The above copyright notice and this permission notice shall be included in all
00014  * copies or substantial portions of the Software.
00015  *
00016  * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017  * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018  * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019  * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00021  * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00022  * SOFTWARE.
00023  */
00024
00033 #ifndef ERRIEZ_OREGON_THN128_TRANSMIT_H_
00034 #define ERRIEZ_OREGON_THN128_TRANSMIT_H_
00035
00036 #ifdef __cplusplus
00037 extern "C" {
00038 #endif
00039
00040 #include <stdio.h>
00041 #include <stdint.h>
00042 #include "ErriezOregonTHN128.h"
00043
00044 void OregonTHN128_TxBegin(uint8_t rfTxPin);
00045 void OregonTHN128_TxRawData(uint32_t rawData);
00046 void OregonTHN128_Transmit(OregonTHN128Data_t *data);
00047
00048 #ifdef __cplusplus
00049 }
00050 #endif
00051
00052 #endif // ERRIEZ_OREGON_THN128_TRANSMIT_H_
```

Index