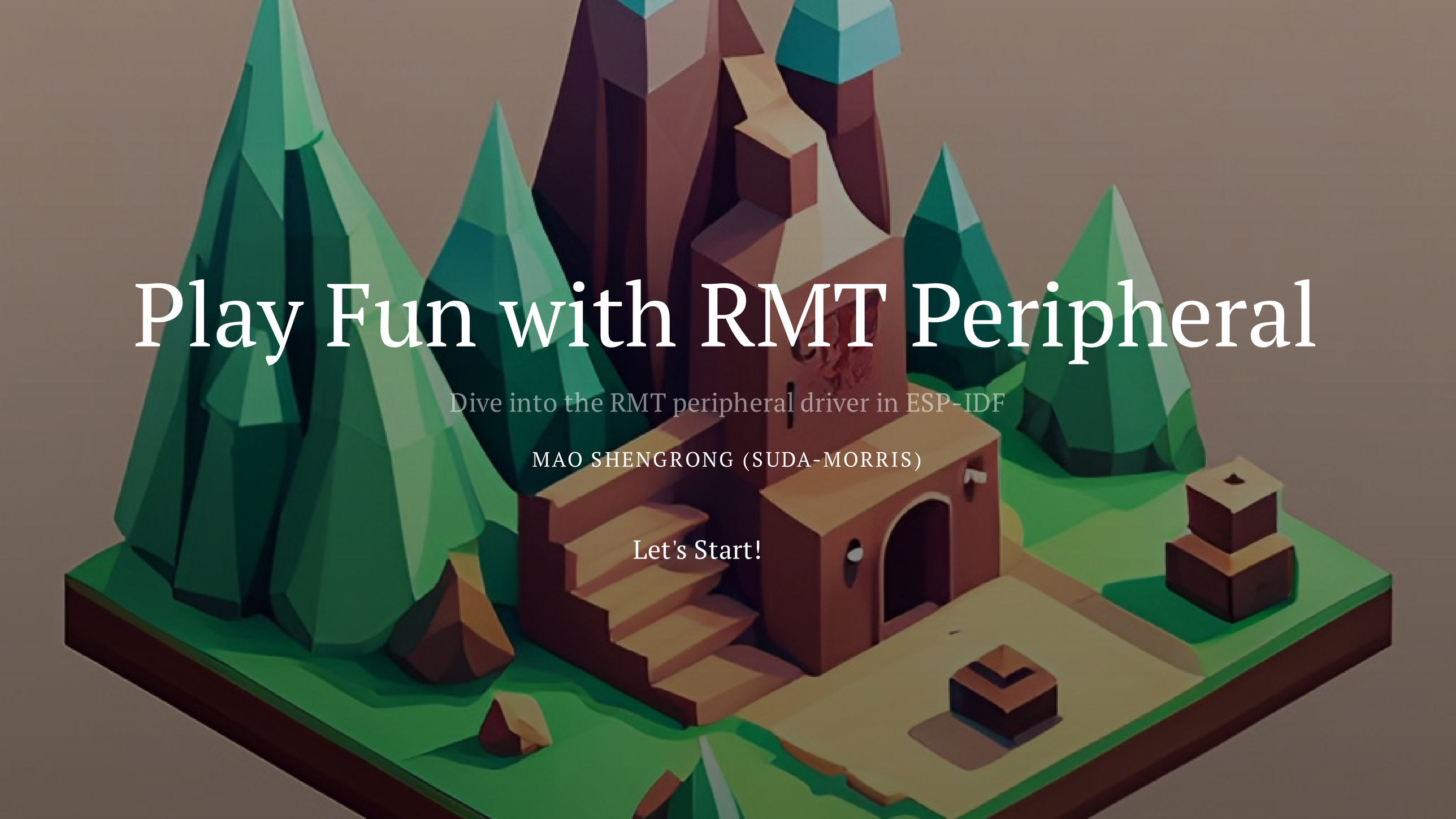


Play Fun with RMT Peripheral

Dive into the RMT peripheral driver in ESP-IDF

MAO SHENGRONG (SUDA-MORRIS)

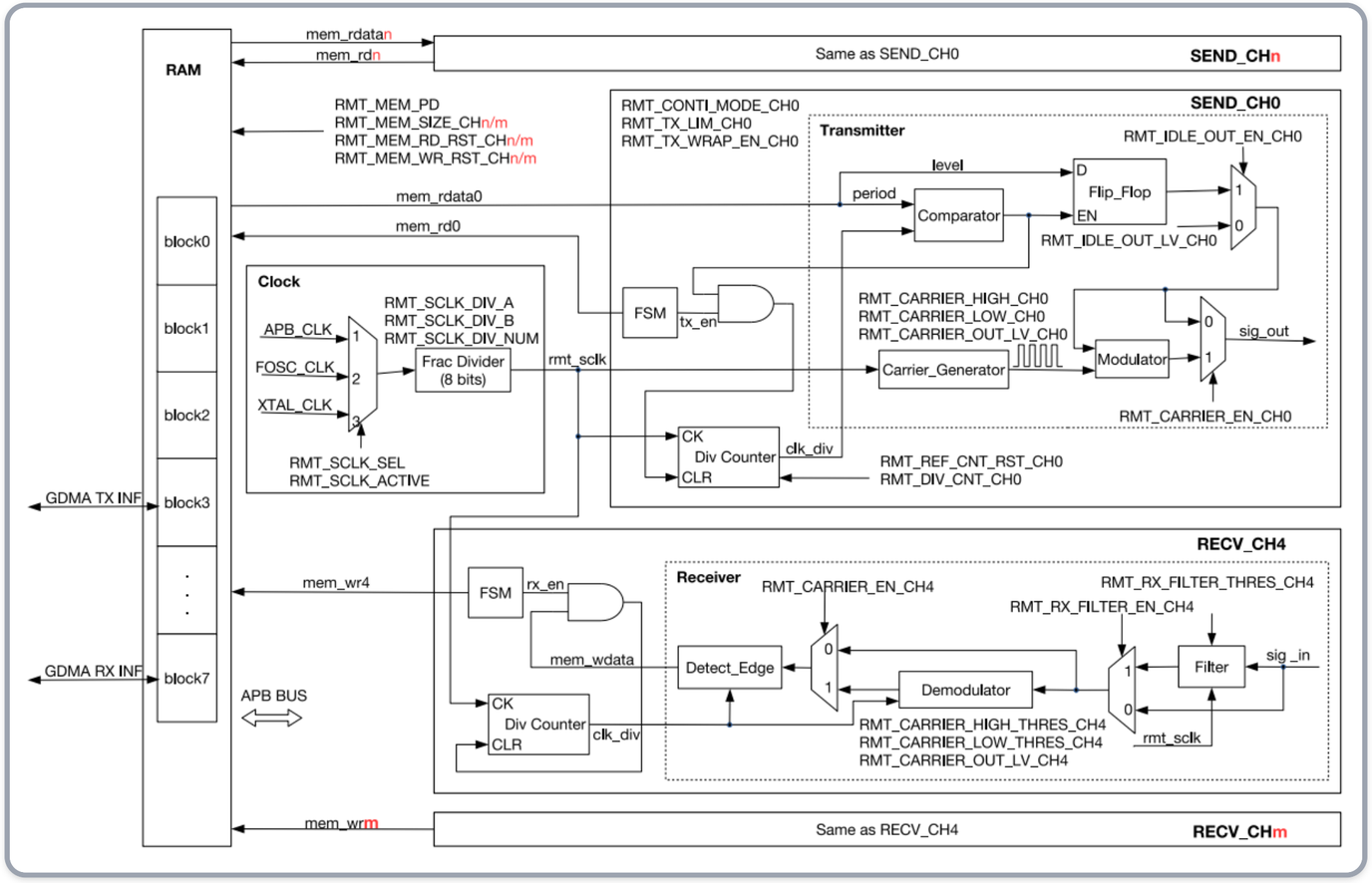
Let's Start!

Overview

RMT was designed for Sending and Receiving *IR Remote* signals, but later it turns out to be a versatile peripheral that can act like a general purpose transceiver.

- 📖 **Quick HW background** - RMT symbol definition, features introduction
- 👷 **Get started with TX channel** - General steps to acquire a TX channel and transmit data
- 👷 **Get started with RX channel** - General steps to acquire a RX channel and receive data
- 🤹 **Dive into RMT encoder** - How to customize your own RMT encoder
- 🎪 **Awesome RMT** - What beautiful examples we have of RMT
- 😀 **Wellknown limitations** - Things that RMT can’t do very well and ways to mitigate
- 🙋 **Q&A** - Common questions? Feature requests? Driver bugs? ...

RMT Architecture [1]



- Each channel has slightly different ability.
- All channels share the same **clock** and **interrupt**.
- Only the last channel has **DMA** support.
- One shared block memory. Each channel can take multiple blocks.

1. ESP32-S3 TRM

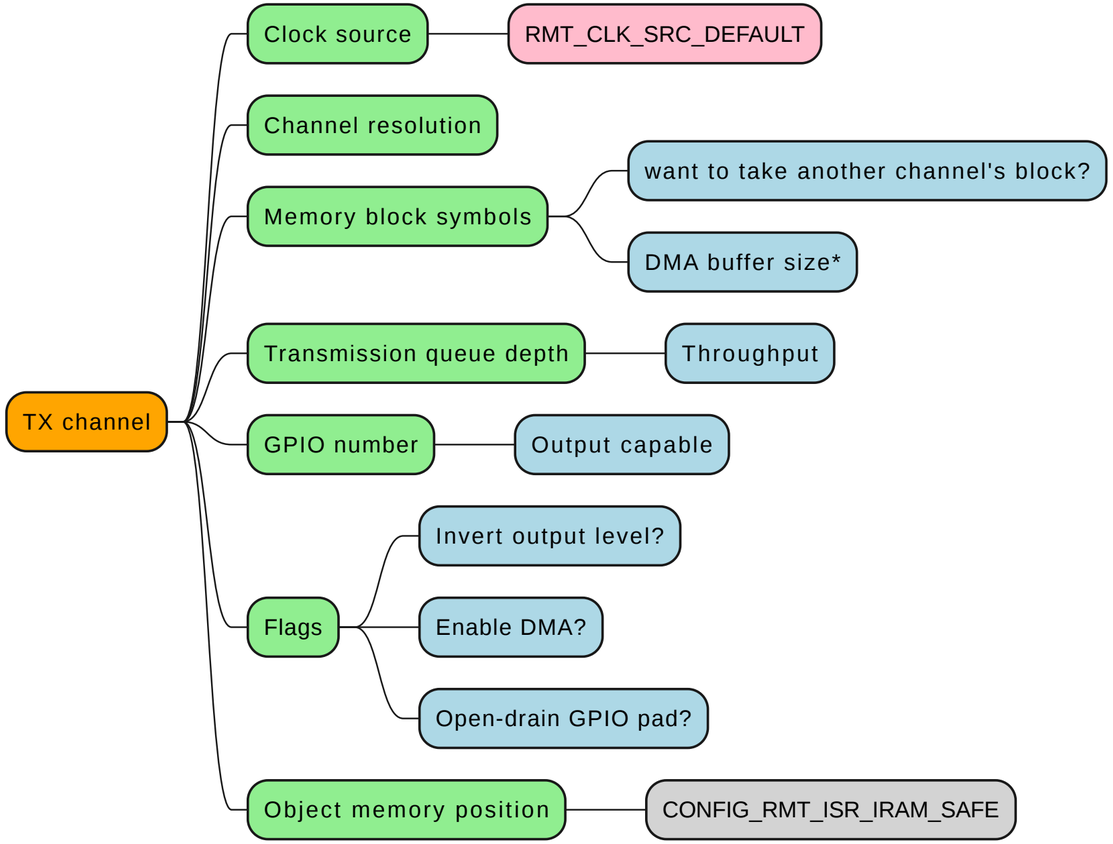
TX Channel Allocation

API

The RMT channels have so many constraints and traps, it's risky to let the user specify the channel directly.

The driver uses a *factory pattern* to help manage all the channel resources and dependencies.

```
rmt_channel_handle_t tx_channel = NULL;
rmt_tx_channel_config_t tx_channel_cfg = {
  .clk_src = RMT_CLK_SRC_DEFAULT,
  .resolution_hz = EXAMPLE_RESOLUTION_HZ,
  .mem_block_symbols = 64,
  .trans_queue_depth = 4,
  .gpio_num = EXAMPLE_IR_TX_GPIO_NUM,
  .flags = {
    .invert_out = false,
    .with_dma = false,
    .io_od_mode = false,
  },
};
rmt_new_tx_channel(&tx_channel_cfg, &tx_channel);
// if CONFIG_RMT_ISR_IRAM_SAFE is enabled,
// the channel object is allocated on SRAM for sure.
```

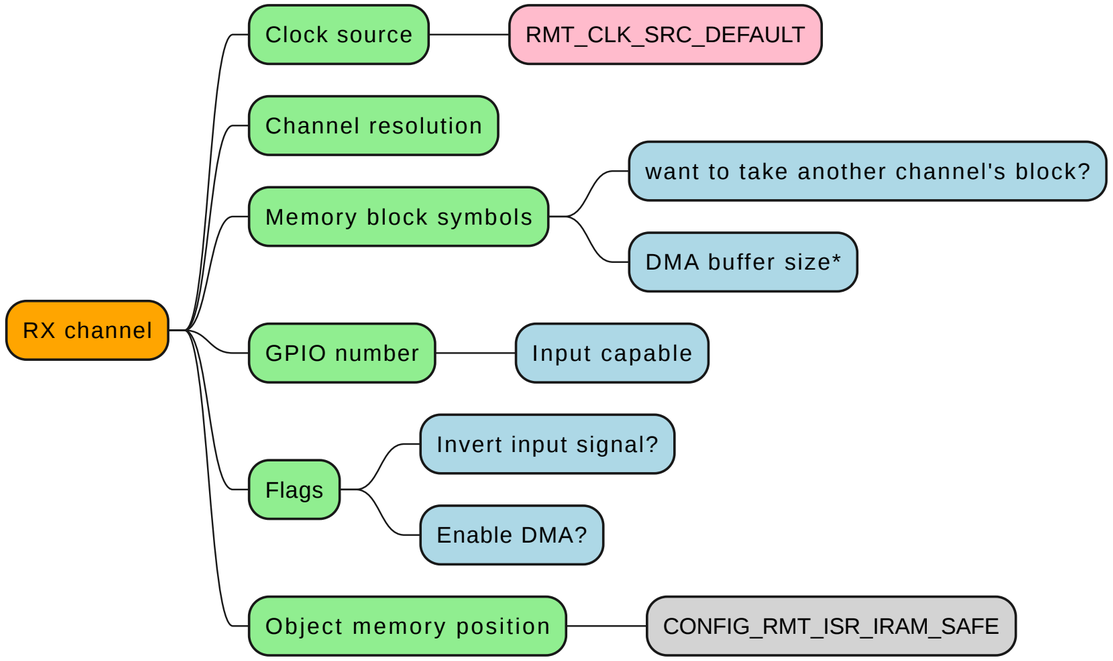


RX Channel Allocation API

The configuration of the RX channel and the TX channel is almost symmetrical.

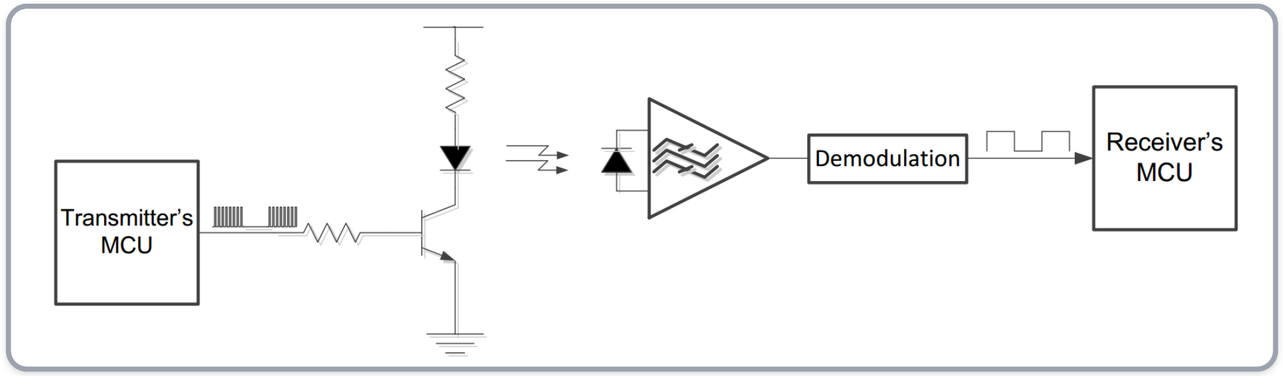
If the TX channels have taken all the memory blocks, the RX channel will fail to allocate, vice versa.

```c
rmt_channel_handle_t rx_channel = NULL;
rmt_rx_channel_config_t rx_channel_cfg = {
  .clk_src = RMT_CLK_SRC_DEFAULT,
  .resolution_hz = EXAMPLE_RESOLUTION_HZ,
  .mem_block_symbols = 64,
  .gpio_num = EXAMPLE_IR_RX_GPIO_NUM,
  .flags = {
    .invert_in = false,
    .with_dma = false,
  },
};
rmt_new_rx_channel(&rx_channel_cfg, &rx_channel);
```



ESP32 & ESP32-S2	Others	DMA
Ping-Pong mode ❎ It determines the maximum RMT symbols in one transaction.	Ping-Pong mode ✅ It sets the memory blocks it will take implicitly	DMA buffer size

Carrier API



- Carrier is applied to the **high** level of the base signal, unless `polarity_active_low` is true
- Carrier shows up only when there's data transfer on the line, unless `always_on` is true

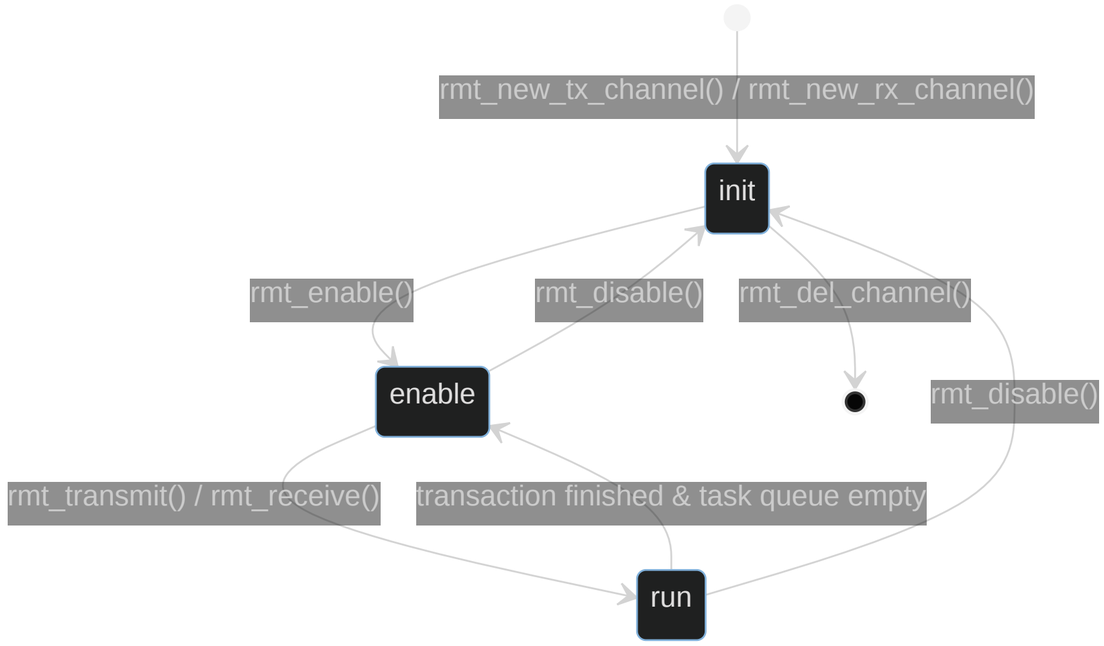
```
rmt_carrier_config_t carrier_cfg = {
  .frequency_hz = 38000,
  .duty_cycle = 0.33,
  .flags = {
    .polarity_active_low = false,
    .always_on = false,
  },
};
rmt_apply_carrier(tx_channel, &carrier_cfg);
```

- Carrier frequency set for the RX channel should be slightly **smaller** than the theoretical value so that the receiver can detect the carrier signal correctly

```
rmt_carrier_config_t carrier_cfg = {
  .frequency_hz = 25000,
  .duty_cycle = 0.33,
};
rmt_apply_carrier(rx_channel, &carrier_cfg);
```

State Transition

Internal



- `rmt_enable` / `rmt_disable` will *acquire* / *release* the power management lock
- When the RMT TX channel is running in an infinite hardware loop, the only way to stop it is `rmt_disable`
- `rmt_enable` can pick a pending transaction from queue and start it

```
esp_err_t rmt_enable(rmt_channel_handle_t channel);
esp_err_t rmt_disable(rmt_channel_handle_t channel);
```

Transmit & Receive API

TX channel Non-blocking

```
rmt_transmit_config_t transmit_config = {
  .loop_count = 0,
  .flags.eot_level = 0,
};
rmt_transmit(led_chan, led_encoder,
             led_strip_pixels, sizeof(led_strip_pixels),
             &transmit_config);
```

- `eot_level` controls the output level after the last symbol is sent
- `led_encoder` is a user customized encoder

`loop_count`	-1	0	N > 0
Description	Infinite loop	Single shot	N times

RX channel Non-blocking

```
rmt_receive_config_t receive_config = {
  .signal_range_min_ns = 1250,
  .signal_range_max_ns = 12000000,
};
rmt_symbol_word_t raw_symbols[64];
rmt_receive(rx_channel, raw_symbols, sizeof(raw_symbols),
            &receive_config);
```

- pulse with duration shorter than `signal_range_min_ns` is **noise**
- pulse with duration longer than `signal_range_max_ns` is **STOP signal**
- truncated the received symbols if the user provided buffer is not big enough

Event Callback API

TX channel

```
static bool example_rmt_tx_done_callback(rmt_channel_handle_
{
  uint32_t *user = (uint32_t *)user_data;
  esp_rom_printf("Transmitted %d symbols\n", edata->num_symb
  return false;
}
rmt_tx_event_callbacks_t cbs = {
  .on_trans_done = example_rmt_tx_done_callback,
};
uint32_t user_data = 0;
rmt_tx_register_event_callbacks(tx_channel, &cbs, &user_data
```

RX channel

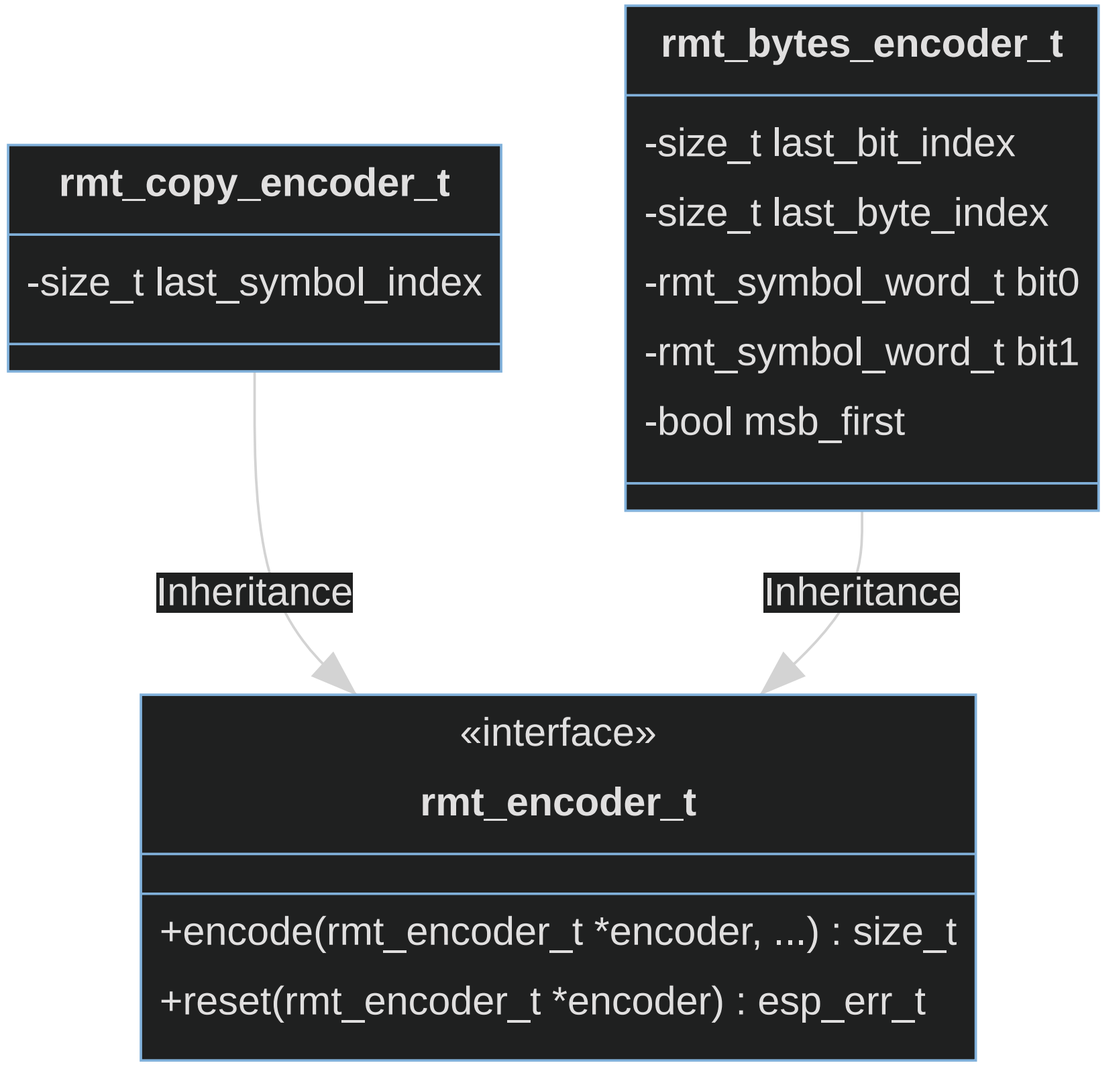
```
static bool example_rmt_rx_done_callback(rmt_channel_handle_
{
  BaseType_t high_task_wakeup = pdFALSE;
  QueueHandle_t receive_queue = (QueueHandle_t)user_data;
  xQueueSendFromISR(receive_queue, edata, &high_task_wakeup)
  return high_task_wakeup == pdTRUE;
}
QueueHandle_t receive_queue = xQueueCreate(1, sizeof(rmt_rx_
rmt_rx_event_callbacks_t cbs = {
  .on_recv_done = example_rmt_rx_done_callback,
};
rmt_rx_register_event_callbacks(rx_channel, &cbs, receive_qu
```

- Callback function is running under the *ISR* context
- Callback function returns `true` to indicate a need of context switch
- if `CONFIG_RMT_ISR_IRAM_SAFE` enabled
  - 👍 Interrupt event won't be differed along with flash operations
  - 👎 Callback function should not access the flash memory

Native Encoder

API

RMT encoder is composable , you can combine any native encoders into a new encoder.

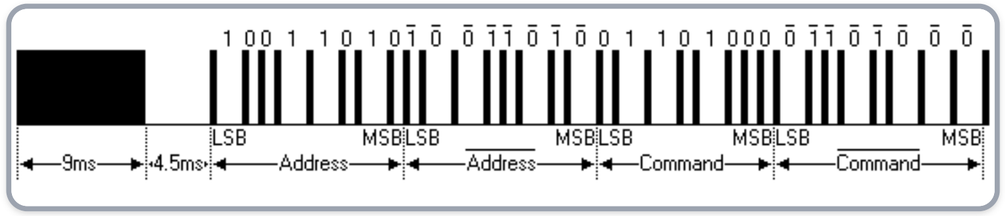



```c
rmt_encoder_handle_t encoder = NULL;
rmt_copy_encoder_config_t copy_encoder_config = {};
rmt_new_copy_encoder(&copy_encoder_config, &encoder);
```

```c
rmt_encoder_handle_t encoder = NULL;
rmt_bytes_encoder_config_t bytes_encoder_config = {
  .bit0 = {
    .level0 = 1,
    .duration0 = 3,
    .level1 = 0,
    .duration1 = 9,
  },
  .bit1 = {
    .level0 = 1,
    .duration0 = 9,
    .level1 = 0,
    .duration1 = 3,
  },
  .flags.msb_first = 1,
};
rmt_new_bytes_encoder(&bytes_encoder_config, &encoder);
```

NEC Encoder

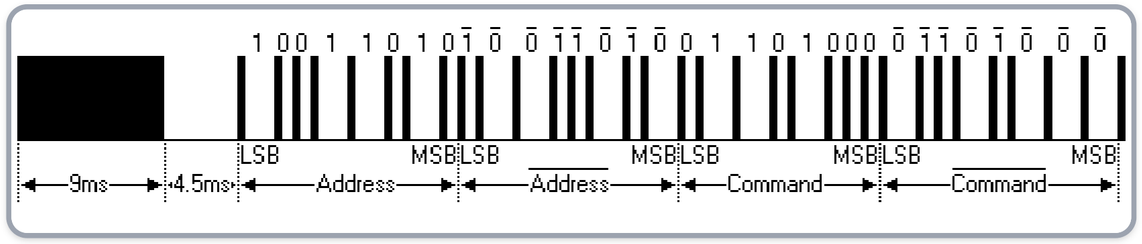
Example

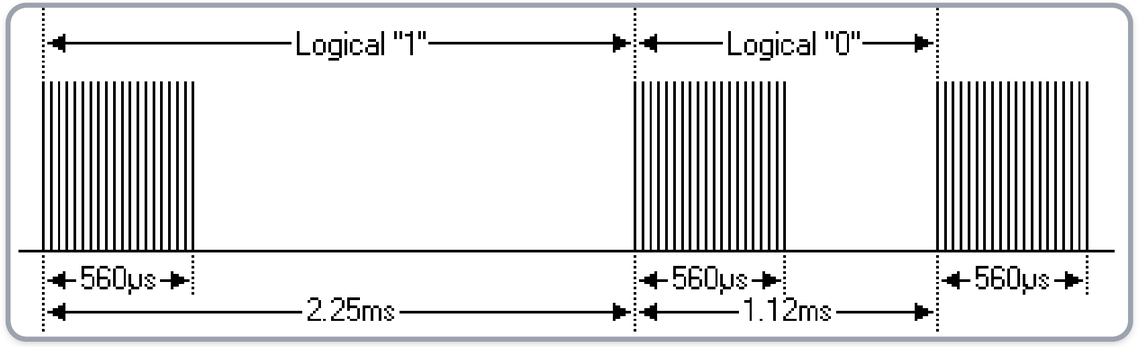



```
typedef struct {
  rmt_encoder_t base;
  rmt_encoder_t *copy_encoder;
  rmt_encoder_t *bytes_encoder;
  rmt_symbol_word_t nec_leading_symbol;
  rmt_symbol_word_t nec_ending_symbol;
  int state;
} rmt_ir_nec_encoder_t;
nec_leading_symbol = (rmt_symbol_word_t) {
    .level0 = 1,
    .duration0 = 9000ULL * rmt_chan_res / 1000000,
    .level1 = 0,
    .duration1 = 4500ULL * rmt_chan_res / 1000000,
};
nec_ending_symbol = (rmt_symbol_word_t) {
    .level0 = 1,
    .duration0 = 560 * rmt_chan_res / 1000000,
    .level1 = 0,
    .duration1 = 0x7FFF,
};
```

NEC Encoder

Example





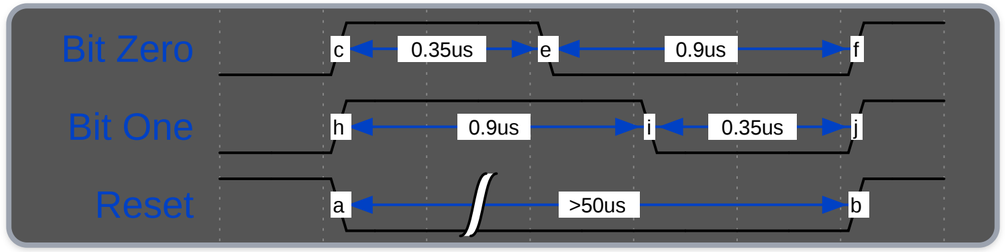

```c
typedef struct {
    uint16_t address;
    uint16_t command;
} ir_nec_scan_code_t;
```

```c
static size_t rmt_encode_ir_nec(rmt_encoder_t *encoder, rmt_channel_handle_t channel,
                                const void *primary_data, size_t data_size,
                                rmt_encode_state_t *ret_state)
{
    rmt_ir_nec_encoder_t *nec_encoder = __containerof(encoder, rmt_ir_nec_encoder_t, base);
    rmt_encode_state_t session_state = RMT_ENCODING_RESET;
    rmt_encode_state_t state = RMT_ENCODING_RESET;
    size_t encoded_symbols = 0;
    ir_nec_scan_code_t *scan_code = (ir_nec_scan_code_t *)primary_data;
    rmt_encoder_handle_t copy_encoder = nec_encoder->copy_encoder;
    rmt_encoder_handle_t bytes_encoder = nec_encoder->bytes_encoder;
    switch (nec_encoder->state) {
```

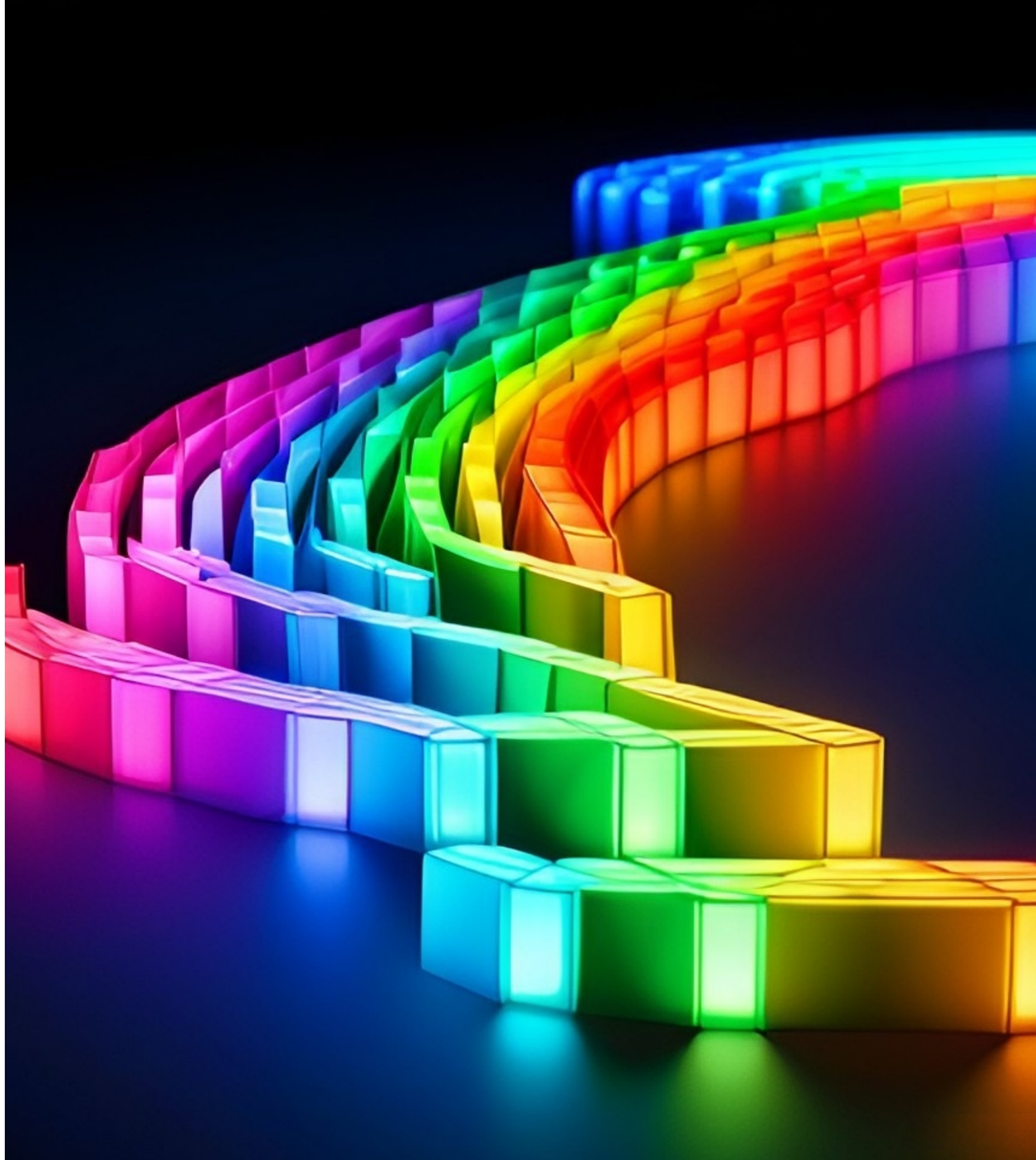
LED Strip

Example

latest version 2.3.1

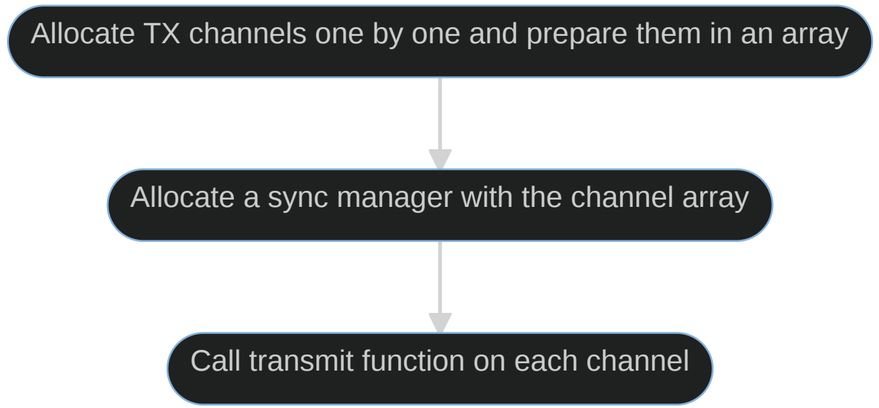



```
static size_t rmt_encode_led_strip(rmt_encoder_t *enc
                                   const void *primar
                                   rmt_encode_state_t
{
    rmt_led_strip_encoder_t *led_encoder = __containe
    rmt_encoder_handle_t bytes_encoder = led_encoder-
    rmt_encoder_handle_t copy_encoder = led_encoder->
    rmt_encode_state_t session_state = RMT_ENCODING_R
    rmt_encode_state_t state = RMT_ENCODING_RESET;
    size_t encoded_symbols = 0;
    switch (led_encoder->state) {
    case 0: // send RGB data
        encoded_symbols += bytes_encoder->encode(byte
        if (session_state & RMT_ENCODING_COMPLETE) {
            led_encoder->state = 1; // switch to next
        }
        if (session_state & RMT_ENCODING_MEM_FULL) {
```



TX Sync API

```
rmt_channel_handle_t tx_channels[TEST_RMT_CHANS];
rmt_sync_manager_handle_t synchro = NULL;
rmt_sync_manager_config_t synchro_config = {
  .tx_channel_array = tx_channels,
  .array_size = TEST_RMT_CHANS,
};
rmt_new_sync_manager(&synchro_config, &synchro);
for (int i = 0; i < TEST_RMT_CHANS; i++) {
  rmt_transmit(tx_channels[i], led_strip_encoders[i],
}
```



> The transmissions won't start until the last managed channel starts the transmission successfully.

Other Examples

Example

Example	Descriptions	Key RMT features used
	D-Shot protocol Electronic Speed Controller	Infinite RMT transmission loop
	Stepper Motor Smooth Controller	Precise number of RMT transmission loop
	1-Wire Sensor: DS18B20	Bind TX and RX channel to the same GPIO

Learn More

- Examples
- TRM
- Programming Guide

Remote Control Transceiver (RMT)

Introduction

The RMT (Remote Control Transceiver) peripheral was designed to act as an infrared transceiver. However, due to the flexibility of its data format, the functionality of RMT can be extended to a versatile and general purpose transceiver. From the perspective of network layering, the RMT hardware contains both physical and data link layer. The physical layer defines the communication media and bit signal representation. The data link layer defines the format of an RMT frame. The minimal data unit in the frame is called **RMT**

Thank You! 

Slides source code can be found on Github