Hands on session 06: Ambient Light control

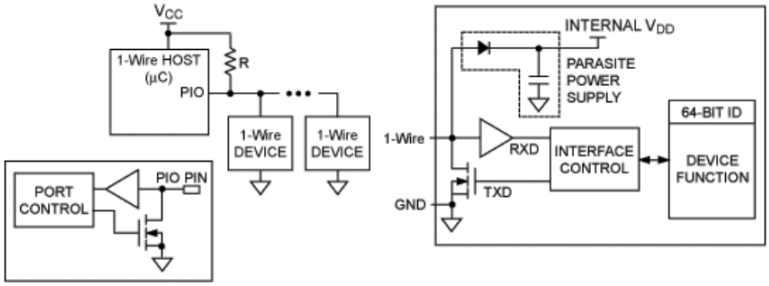
Session Introduction: In this session we will create an ambient light control using capacitive touch feature of ESP32 and WS2812B.

Session Objectives:

- **To understand single wire protocols**
- **To understand the working of WS2812 addressable RGB LED's**
- **To understand capacitive touch sensor**
- **To make ambient light control using touch**

Single wire protocols

The basis of single wire technology is a serial protocol using a single data line plus ground reference for communication. A 1-Wire master initiates and controls the communication with one or more 1-Wire slave devices on the 1-Wire bus. Each 1-Wire slave device has a unique, unalterable, factory-programmed, 64-bit identification number (ID), which serves as device address on the 1-Wire bus. The 8-bit family code, a subset of the 64-bit ID, identifies the device type and functionality.



The 1-Wire master/slave configuration uses a single data line plus ground reference.

1-Wire is a voltage-based digital system that works with two contacts, data and ground, for half- duplex bidirectional communication. Compared to other serial communication systems such as I^2C or SPI, 1-Wire devices are designed for use in a momentary contact environment. Either disconnecting from the 1-Wire bus or a loss of contact puts the 1-Wire slaves into a defined reset state. When the voltage returns, the slaves wake up and signal their presence. With only one contact to protect, the built-in ESD protection of 1-Wire devices is extremely high. With two contacts, 1-Wire devices are the most economical way to add electronic functionality to nonelectronic objects for identification, authentication, and delivery of calibration data or manufacturing information.

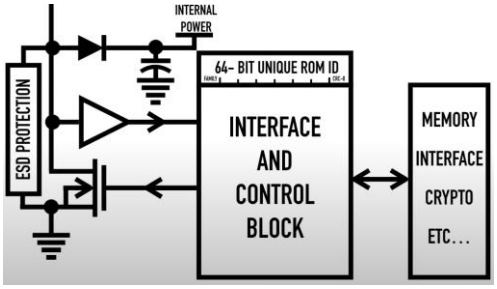
A one wire interface has just one wire and ground. That one wire connects to an open drain bus means any device on bus can pull the voltage on bus to the ground but no device can drive the bus high.

Every one bus consists of a passive pull up resistor. There is a FET switch in every single wire device to pull the bus down when there is time to send the data and there is a buffer to square up the signals from the io line

For power a rectifier is included in a small internal capacitor and this is called parasite power and this is used to run the internal logic of the one wire device even when another citizen on the network is pulling the I/O line low.

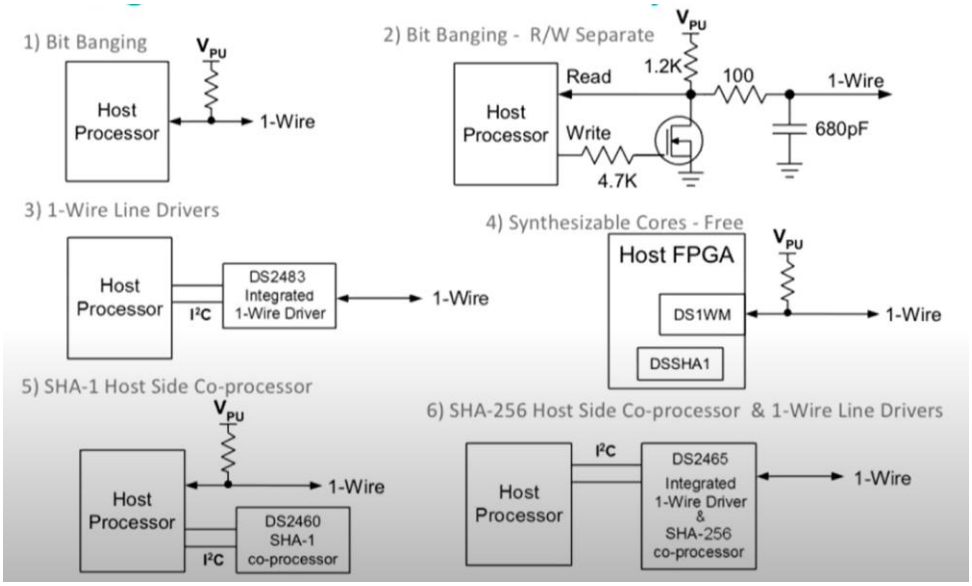
The sequence of bits coming from the host is presented to an interface and control block. This block is present in every one wire device.

The interface and control block includes a 64-bit identifier that's unique to every one wire device.



A one wire device

There is various implementation of one wire networks the simplest implementation is to use simple spare open drain microcontroller port pin with an external pullup resistor to bit bang the one wire line.



Some sensors/actuators working on single wire protocols

DS18S20

DHT 11

WS2812

Bit Banging

In computer engineering and electrical engineering, **bit banging** is slang for any method of data transmission that employs software as a substitute for dedicated hardware to generate transmitted signals or process received signals. Software directly sets and samples the states of GPIOs, and is responsible for meeting all timing requirements and protocol sequencing of the signals. In contrast to bit banging, dedicated hardware (e.g., UART, SPI interface) satisfies these requirements and, if necessary, provides a data buffer to relax software timing requirements. Bit banging can be implemented at very low cost, and is commonly used in embedded systems.

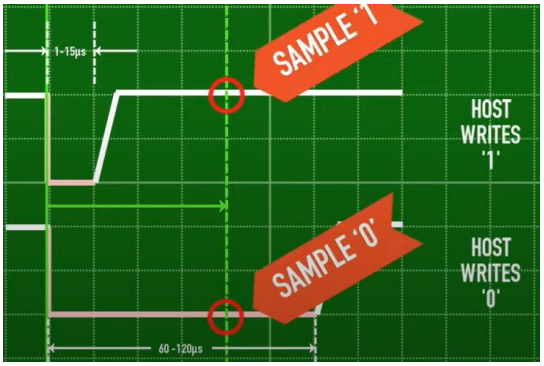
Bit banging allows a device to implement different protocols with minimal or no hardware changes. In some cases, bit banging is made feasible by newer, faster processors because more recent hardware operates much more quickly than hardware did when standard communications protocols were created.

Sending and receiving data using this protocol:

To send a one bit the host pulls down the I/O line and keeps it low for 1-15 micro seconds and then releases it for the 65 microseconds of bit period

To send the zero bit the host pulls down the I/O line for 60-120 micro seconds and then leaves the line high for at least 5 microseconds

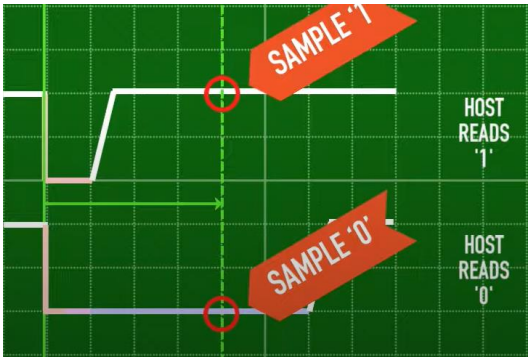
The falling edge of I/O line starts the timer and a few microseconds later the one wired device samples the line.



When the host wants **to read** some thing from one wired device

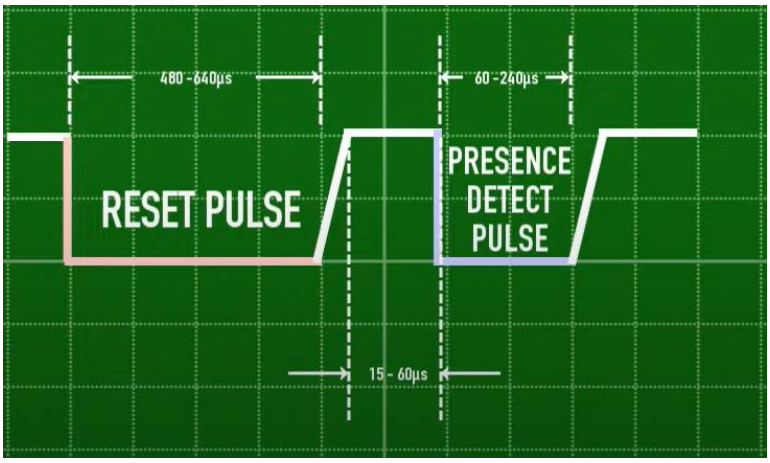
The host first sends the one and if the device wants to send back one then the device allows the line to float up to the high level and when host samples the line a few microseconds later it reads the one

If the device wants to send a zero as soon as it sees the line falling to low it starts pulling the line and a few microseconds later when microcontroller samples the line it reads 0.



To start transaction the host holds the I/O line down for 480 to 640 microseconds and it is called a reset pulse and that's how the host gets the attention of all the one wired devices on the line.

When any one wired device sees a reset pulse it answers by driving the I/O line low for 60-240 microseconds and it is called the presence detect pulse.



WS2812

Datasheet WS2812: https://cdn-shop.adafruit.com/datasheets/WS2812.pdf

WS2812 Protocol

Each separate red, green, and blue LED in a single WS2812 unit is set up to shine at 256 brightness levels, indicated by an 8-bit binary sequence set from 0 to 255. When combined, each LED unit requires three sets of eight brightness bits, or 24 bits, of information for full control.

1. A **microcontroller** transmits this sequence of eight green bits, eight red bits, and eight blue bits to the first LED in the series.

2. When multiple LEDs are present, the data sequence that controls the second LED starts directly after the first with green, red, and blue data. The sequence continues in that pattern until it illuminates every LED present.

3. The first LED takes in information for the entire chain of LEDs, then passes the same data along without the sequence it applied to itself, transforming the second LED into the first component on the list (as far as it knows).

4. This "new number one" LED unit continues passing information along until there are no more binary LED sequences left.

To work with WS2812 we are using [Freenove_WS2812_Lib_for_ESP32](#) library.

To use this library

1) Create its object
```
Freenove_ESP32_WS2812 strip = Freenove_ESP32_WS2812(LEDS_COUNT, LEDS_PIN, CHANNEL, TYPE_GRB);
```

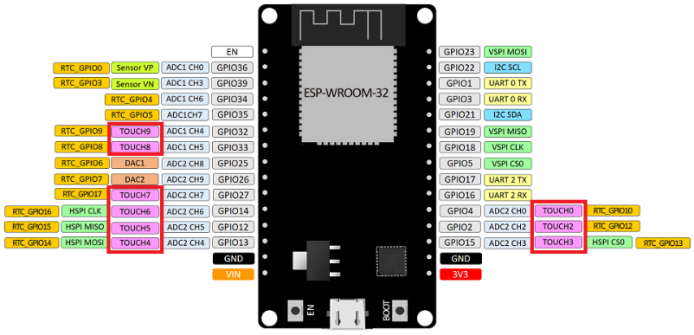
2) Initialization data, ready for communication.
strip.begin()

3) Send colour data and display it immediately.
strip.setLedColor(id, colour); //colour value. egg, 0xFF0000 is RED colour.
strip.setLedColor(id, r, g, b); //colour value. 0-255

4) A simple colour picker
strip.Wheel(i) // i: 0-255.

Touch Sensor

The ESP32 has 10 capacitive touch GPIOs. These GPIOs can sense variations in anything that holds an electrical charge, like the human skin. So, they can detect variations induced when touching the GPIOs with a finger.



These pins can be easily integrated into capacitive pads, and replace mechanical buttons.

Here we pin no. 14 is being used for sensing touch.

touchRead()

Reading the touch sensor is straightforward. In the Arduino IDE, you use the touchRead() function, that accepts as argument, the GPIO you want to read.

touchRead(GPIO); // in our case GPIO = 14

Ambient light with touch controller

Connections

WS2812

- **VCC-> 5v**
- **GND->gnd**
- **DIN -> GPIO 33**
- **DOUT -> WS2812(next)**

Touch pad

- **GPIO 14**