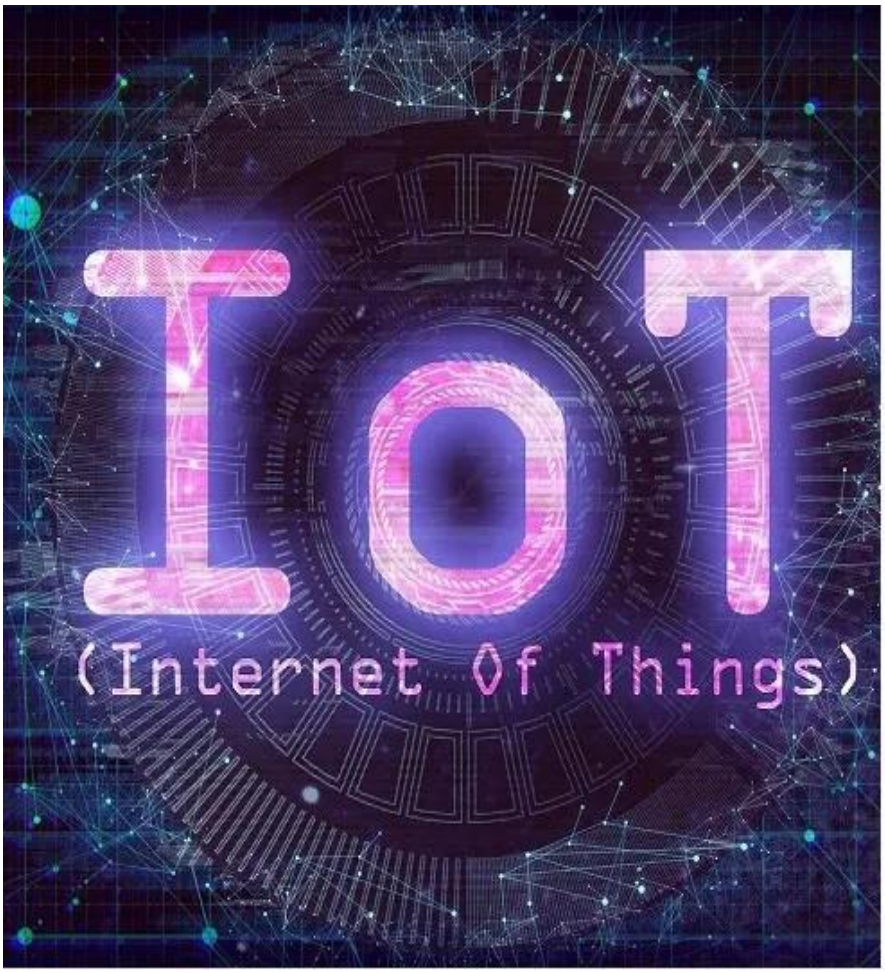
# IoT-based KY-010 PHOTO INTERRUPTER MODULE




IoT Levels & Deployment Templates

**PRESENTED BY**

**VEERESH PATIL**

**NOVEMBER-2023**

**TABLE OF CONTENT**

## 1.PROJECT SUMMARY

The KY-010 Photo Interrupter Module, a versatile sensor module widely employed in electronics projects, operates on the principle of infrared light interruption. Comprising an infrared emitter and photodetector strategically placed with a gap in between, the module excels in object detection. The emitter emits infrared light, and the photodetector registers interruptions caused by objects passing through the gap. With straightforward connections, including VCC, GND, and DO pins, the module seamlessly integrates with microcontrollers like Arduino. Its applications span diverse projects, finding utility in scenarios requiring the detection of objects or rotational motion. The module's pins facilitate easy incorporation into circuits, with the digital output pin serving as a key indicator of the presence or absence of an object.

A typical project involves interfacing the KY-010 with an Arduino, where the microcontroller reads the digital output to discern the status of the detection gap. This information can then be utilized to trigger specific actions, such as activating LEDs or displaying messages. The ease of programming and integration makes the KY-010 suitable for various applications, from automated gate systems to line-following robots. Its affordability and widespread availability contribute to its popularity among hobbyists and students exploring the realms of electronics.

Beyond its foundational use in object detection, the KY-010 Photo Interrupter Module has found its way into educational settings, where it serves as a hands-on tool for teaching principles of sensors, light detection, and microcontroller interfacing. Its adaptability allows for creative exploration, enabling users to delve into the intricacies of electronics while providing a tangible and practical means for understanding concepts like interruption-based sensing.

However, it's crucial to note the limitations of the module, such as its limited detection range and susceptibility to ambient light interference. Despite these constraints, the KY-010 remains a go-to choice for introductory projects, offering an accessible entry point for individuals interested in the intersection of sensors, microcontrollers, and practical applications in electronics. In summary, the KY-010 Photo Interrupter Module stands as an exemplar of simplicity, effectiveness, and versatility in the realm of electronic sensors, fostering exploration and innovation in a wide array of projects.

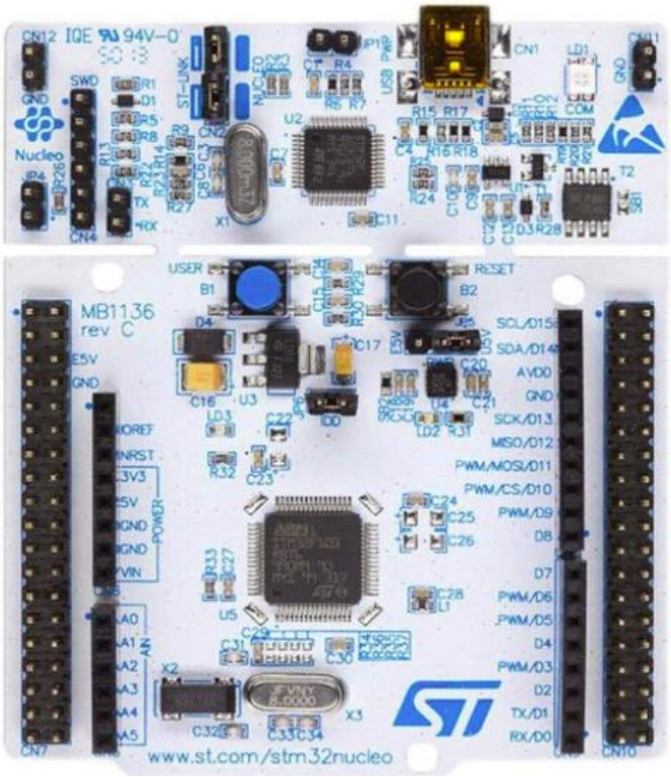
## 2.KEY FEATURES OF THE PROJECT

- Sensor Initialization with Timers.
- UART Communication based on Interrupt.
- Common function for UART receive data parsing.
- Timer based delay.
- STM32 Application is Master and configured with W10 (Wi-Fi) module accordingly with AT commands.
- Data is pushed to the MQTT server.
- When the data is transmitted into rugged-board from stm32f446re communicate between the two board and tranmit the data.
- Object dected(ON/OFF) data is pushed in every time when object is interrupt.

## 3.HARDWARE USED AND THERE SPECIFICATION

### STM32F446RE microcontroller

The STM32F446RE is a high-performance microcontroller based on the ARM Cortex- M4 processor. It has a number of features that make it well-suited for a variety of applications, including:




100 MHz CPU clock speed
128 KB of RAM
512 KB of Flash memory
Floating point unit (FPU)
11 general-purpose timers
13 communication interfaces USB OTG RTC

### RUGGEDBOARD

RuggedBoard - A5D2x is an Single Board Computer providing as easy migration path from Microcontroller to Microprocessor. RuggedBoard is enabled with industry Standard Yocto Build Embedded Linux platform and open source libraries for industrial application development.RuggedBoard is an Open source Industrial single board computer powered by ARM Cortex-A5 SoC @500 MHz, implemented with the finest platform for rapid prototyping. The usage of System On Module over a System On Chip is the most rapid way to achieve time to market, curtail development risks for product quantities ranging from a few hundred to thousands.

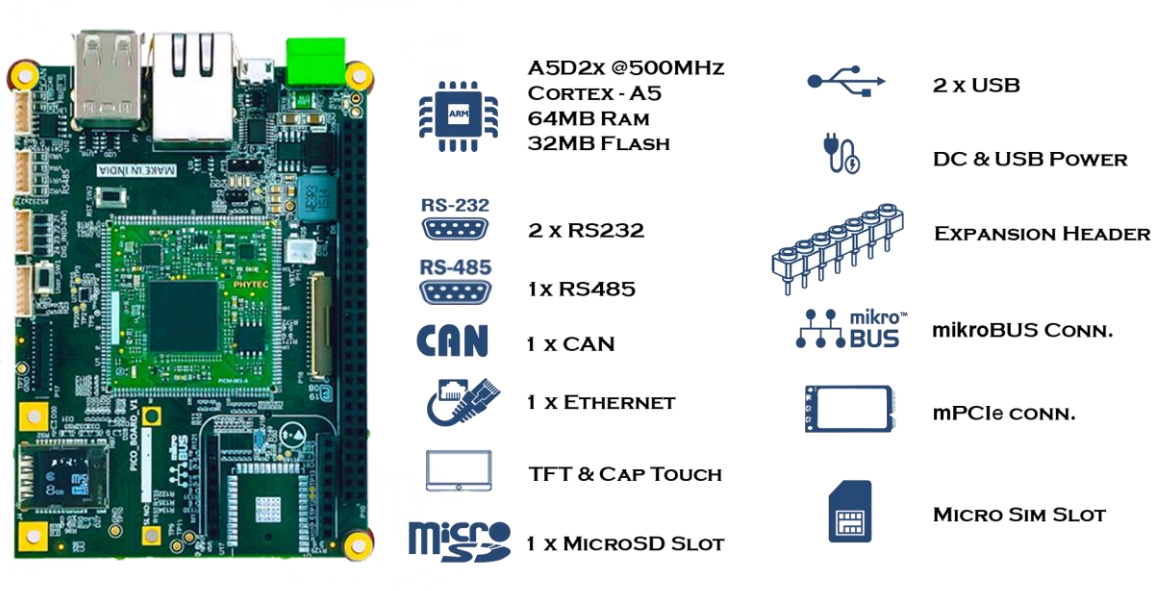
RuggedBoard- A5D2x consists of Multiple Interfaces such as Ethernet, RS232, CAN, RS485, Digital Input and Digital Output with optically isolated, Standard MikroBus header for Add-On Sensors, Actuators and Multiple Wireless Modules such as ZigBee, LoRa, Bluetooth etc. mPCIe connector with USB interface used for Cloud Connectivity modules 3G, 4G, NB-IoT, WiFi. Expansion header with GPIO, UART, I2C, SPI, PWR etc.

**RuggedBoard - A5D2x Specification:**

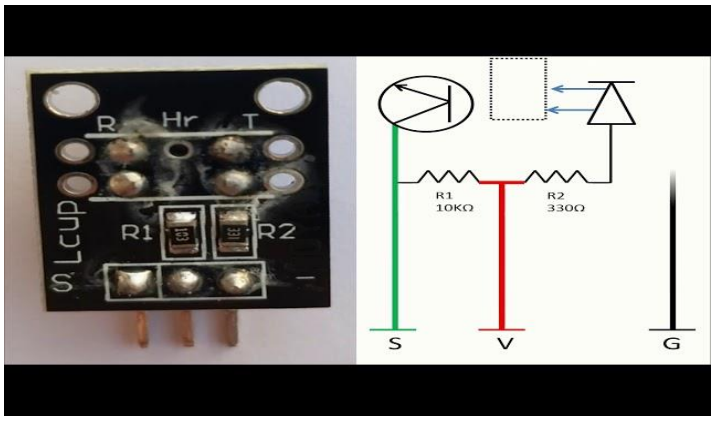
System On Module
SOC Microchip ATSAMA5d2x Cortex-A5
Frequency 500MHz
RAM 64 MB DDR3
Flash 32 MB NOR flash
SD Card SD Card Upto 32 GB

Industrial Interface
RS232 2x RS232
USB 2 x USB*(1x Muxed with mPCIe)
Digital Input 4x DIN (Isolated ~ 24V)
Digital Output 4x DOUT (Isolated ~ 24V)
RS485 1xRs485
CAN 1xCAN
Internet Access
Ethernet 1 x Ethernet 10/100
Wi-Fi/BT Optional on Board Wi-Fi/BT
SIM Card 1 x SIM Slot (for mPCIe Based GSM Module)

Add-On Module Interfaces
Mikro-BUS Standard Mikro-BUS
mPCIe 1 x mPCIe* (Internally USB Signals is used)
Expansion Header SPI, I2C, UART, PWM, GPIO,ADC
Power
Input Power DC +5V or Micro USB Supply
Temperature Range - 40°to + 85°C
Optional Accessories
Accessories Set Micro USB Cable, Ethernet Cable, Power Adapter 5V/3A



**KY-010 LIGHT BLOCK SENSOR**




1.**Electrical Characteristics:**

- **Operating Voltage:** Typically operates at 3.3V or 5V.
- **Current Consumption:** Specifies the current drawn by the module.

2. **Optical Characteristics:**

- **Emitter Type:** Infrared (IR) LED.
- **Detection Method:** Phototransistor or photodiode.
- **Wavelength of IR Emitter:** The wavelength of the emitted infrared light.

3. **Mechanical Characteristics:**

- **Dimensions:** Physical dimensions of the module.
- **Gap Width:** Distance between the IR emitter and the photodetector.

4. **Output Characteristics:**

- **Output Type:** Digital (typically HIGH or LOW) or analog voltage.
- **Output Signal Level:** Voltage levels for HIGH and LOW states.
- **Response Time:** Time taken for the module to respond to changes in the detection status.

5. **Environmental Characteristics:**

- **Operating Temperature Range:** The range of temperatures within which the module is designed to function.
- **Ambient Light Sensitivity:** Information about how ambient light affects the sensor's performance.

6. **Connection Details:**

- **Pin Configuration:** Pinout details, including VCC, GND, and signal pins.
- **Pin Compatibility:** Information on how to connect the module to microcontrollers or other systems.

7. **Additional Features:**
    - **Built-in Comparator:** Some modules may include a comparator for processing analog signals and producing a digital output.
    - **Indicator LEDs:** Presence of LEDs on the module for indicating the status of detection.
8. **Application Notes:**
    - **Guidelines for Use:** Recommendations for optimum usage.
    - **Typical Applications:** Suggestions for applications where the module can be effectively utilized.

**OPERATION**:

The KY-010 Photo Interrupter Module operates based on the interruption of an infrared (IR) light beam.

1. **Emitter and Detector Setup:**
    - The module consists of two main components: an infrared emitter (usually an IR LED) and a photodetector (such as a phototransistor or photodiode).
    - The emitter continuously emits infrared light across a small gap to the photodetector.
2. **Normal State (No Object Detected):**
    - In the absence of any object, the emitted infrared light travels freely from the emitter to the photodetector without interruption.
    - The photodetector receives a consistent amount of infrared light, and the module remains in a quiescent state.
3. **Object Interruption:**
    - When an object is introduced into the gap between the emitter and the photodetector, it physically blocks the path of the infrared light.
    - The interruption of the light causes a change in the output of the photodetector.
4. **Output Change:**
    - The change in the amount of received infrared light results in a change in the electrical output from the photodetector.
    - The module typically has a digital output pin (DO) that provides a signal indicating the presence or absence of an object. The signal can be HIGH or LOW depending on the module's design.
5. **Microcontroller Interface (Optional):**
    - The module is often connected to a microcontroller, such as an Arduino, to process and utilize the output signal.

- In a simple project, the microcontroller can read the digital output and perform actions based on the detection status. For example, it may activate an LED, display a message, or trigger other components.

6. **Applications:**

    - The KY-010 Photo Interrupter Module is commonly used in applications requiring object detection or counting. For instance, it can be employed in line-following robots, rotary encoder systems, or as part of automated systems where the presence or absence of an object needs to be monitored.

## 4.WHY INTERRUPT BASED UART COMMUNICATION

The advantages of interrupt-based UART communication over other methods:

- Improved efficiency: The CPU is not constantly polling the UART status, so it can be used for other tasks. This can improve the efficiency of the system and reduce power consumption.
- Reduced latency: Interrupt-based communication can be faster than polling, as the CPU is only interrupted when data is ready to be sent or received. This can improve the responsiveness of the system.
- Better multitasking: Interrupt-based communication allows the CPU to handle multiple UART events simultaneously. This can be useful for applications that require frequent and asynchronous serial communication.
- Here are some of the disadvantages of interrupt-based UART communication:
- More complex code: Interrupt-based communication is more complex to implement than polling. This is because the programmer needs to write code to handle the interrupts.
- More overhead: Interrupt-based communication can have more overhead than polling. This is because the CPU needs to save and restore its state when it is interrupted.
- Overall, interrupt-based UART communication is a more efficient and responsive way to communicate with serial devices. However, it is more complex to implement and can have more overhead.
- Here are some examples of applications where interrupt-based UART communication would be a good choice.
- Sensor networks: Sensor networks often need to communicate with each other or with a central server. Interrupt-based communication can be used to improve the efficiency and responsiveness of these networks.
- Wireless modules: Wireless modules often use UART to communicate with the host microcontroller. Interrupt-based communication can be used to improve the performance of these modules.
- Real-time systems: Real-time systems often need to communicate with other devices in a timely manner. Interrupt-based communication can be used to ensure that these communications are not missed.

**STAGE**

**STAGE:1**

The STM32F446RE with KY-010 is used to communicate with the module and board. The sensor uses a beam of light between the emitter and detector to check if the path between both is being blocked by an opaque object.

**STAGE:2**
The STM32F446RE with KY-010,W10 is used to transmit the data into the MQTT server to Publish the data in the Right-tech.

**STAGE:3**
The Ruggedboard with STM32F446RE is used to transmit the data and get the KY-010 value in the ruggedboard minicom.

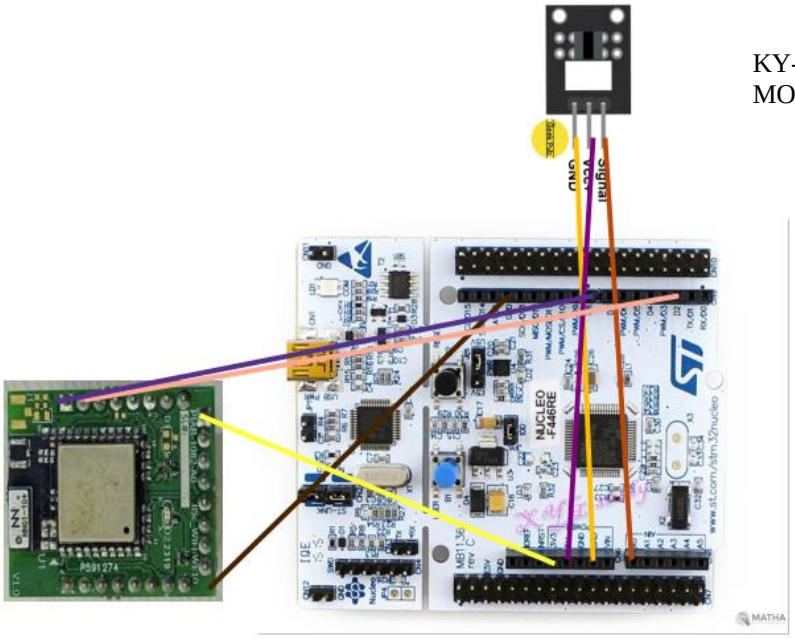
**STAGE:4**
The Ruggedboard with STM32F446RE is used to transmit the data and get the KY-010 value in the ruggedboard minicom and pass the value into the Right-tech by using W10 module is connected with the Rugged board.

## 5. CONNECTION DIAGRAM

**STAGE:2**

KY-010 PHOTO INTERRUPTER MODULE

W10 WIFI



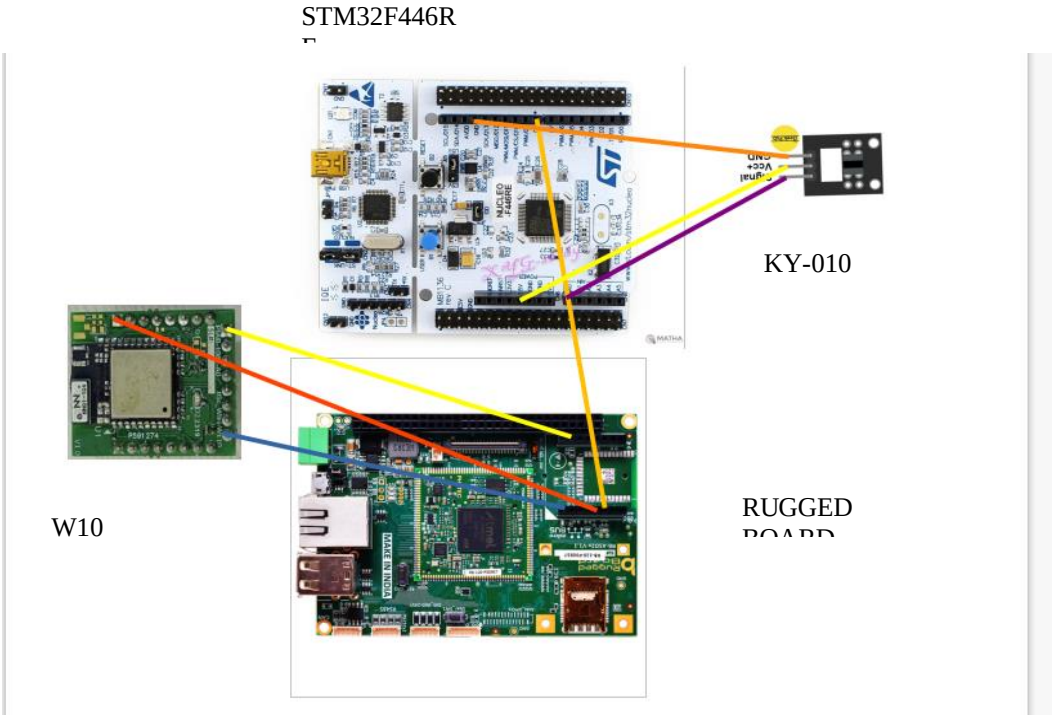

STM32F446R

PIN CONNECTION:

| PIN | PIN NUMBER | COMPONENTS |
|---|---|---|
| SIGNAL PIN | PA0 | KY-010 |

| | | |
|---|---|---|
| GROUND | GND | KY-010 & W10 WIFI |
| VCC | VCC +5V | KY-010 |
| VCC | VCC +3.3V | W10 WIFI |
| Tx | PA9 | W10 WIFI |
| Rx | PA10 | W10 WIFI |

**STAGE:4**




**STM32F446RE WITH KY-010:**

| PIN | PIN NUMBER | COMPONENTS |
|---|---|---|
| SIGNAL PIN | PA0 | KY-010 |
| GROUND | GND | KY-010 |
| VCC | VCC +5V | KY-010 |

**RUGGED-BOARD WITH W10 WIFI:**

| PIN | PIN NUMBER | COMPONENTS |
|---|---|---|

| | | |
|---|---|---|
| VCC | VCC 3.3V | W10 WIFI |
| GROUND | GND | W10 WIFI |
| Tx | Rx | RB to W10 WIFI |
| Tx | Rx | STM32 to RB |

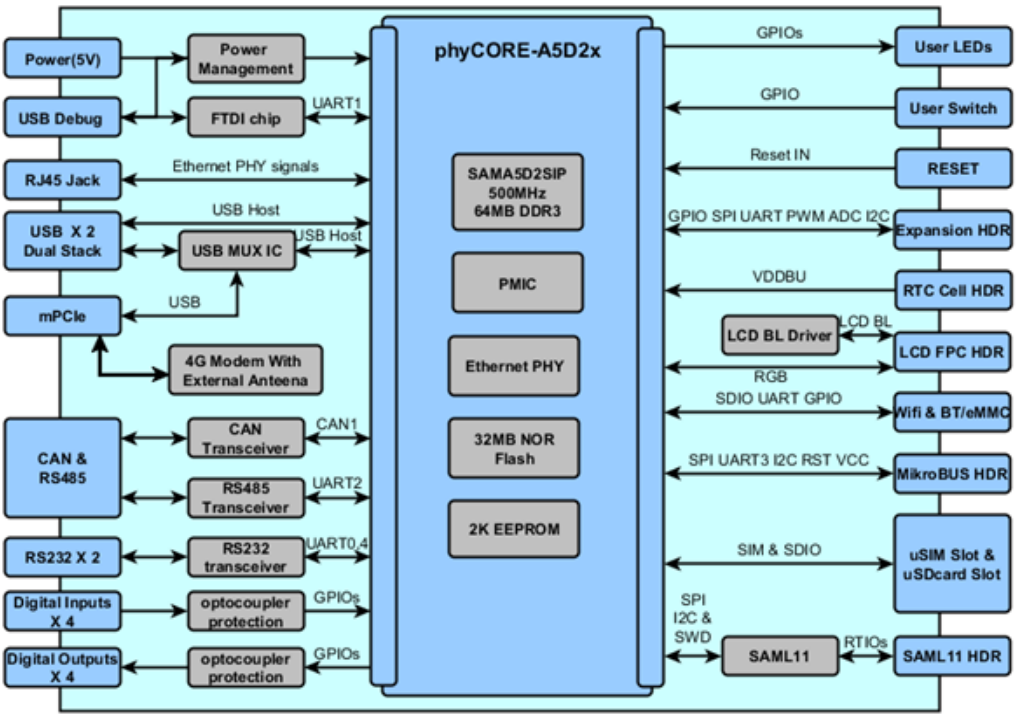
## 6. KY-010 SENSOR

The KY-010 Photo Interrupter Module serves as a fundamental component in electronic projects designed for object detection. Comprising an infrared emitter and a photodetector arranged with a gap between them, the module operates by emitting infrared light and detecting interruptions caused by the presence of an object within the gap. With straightforward connections involving VCC, GND, and DO pins, it easily integrates with microcontrollers like Arduino. This module finds applications in projects requiring object detection, and when coupled with an Arduino, it enables the development of simple yet effective systems. An illustrative project involves connecting the KY-010 to an Arduino, reading its digital output to ascertain the presence or absence of an object, and subsequently triggering actions, such as displaying messages or activating LEDs. Its versatility, coupled with ease of use, renders the KY-010 Photo Interrupter Module a popular choice among hobbyists and students for various electronic applications.

## 7.RUGGED BOARD

The RuggedBoard for phyCORE-A5D2x is a SIP (System in Peripheral) which is a low-cost, feature-rich software development platform supporting the Microchip's A5D2x microprocessor. Moreover, due to the numerous standard interfaces the RuggedBoard A5D2x can serve as bedrock for your application. At the core of the RuggedBoard is the phyCORE-A5D2x System On Module (SOM) in a direct solder form factor, containing the processor, Flash, power regulation, supervision, transceivers, and other core functions required to support the A5D2x processor. Surrounding the SOM is the RuggedBoard carrier board, adding power input, buttons, connectors, signal breakout, Ethernet and mikro-BUS connectivity amongst other things.

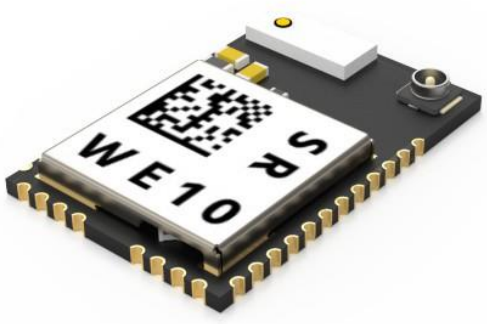
This RuggedBoardoffers an ultra-low cost Single Board Computer for the A5D2x processor, while maintaining most of the advantages of the SOM concept. Adding the phyCORE-A5D2x SOM into your own design is as simple as ordering the connector version and making use of our RuggedBoard Carrier Board reference schematics.




**The RuggedBoard has the following features**

1 x Ethernet
2 x RS-232
1 x RS-485 (Isolated)
1 x CAN
4x DIN (Isolated)
4x DOUT
1 x LVDS Display
1 x Micro SD
1 x SIM
2 x USB 2.0
1 x mikroBUS
1 x 60 PIN Expansion Headers

**8.W10 WiFi**



The W10 WiFi module is a low-cost, easy-to-use WiFi module that can be used to connect IoT devices to the internet. The module has a built-in TCP/IP stack, so it can be easily connected to a variety of IoT platforms. The module also has a number of other features, such as:100mW transmit power 11Mbps data rate,802.11 b/g/n compatibility Integrated antenna.

**9.CODE SNIPPET TO WIFI MODULE INITIALIZATION AND CONNECTION WITH INTERRUPT-BASED UART COMMANDS**

```
void WE10_Init ()
{
        char buffer[128];
        /********* CMD+RESET ********/
        //memset(&buffer[0],0x00,strlen(buffer));
        sprintf (&buffer[0], "CMD+RESET\r\n");
        HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);



        HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);


        /*********  CMD+WIFIMODE=1  ********/
        //memset(&buffer[0],0x00,strlen(buffer));
        sprintf (&buffer[0], "CMD+WIFIMODE=1\r\n");
        HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);

        HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);


        /********* CMD+CONTOAP=SSID,PASSWD ********/
        //memset(&buffer[0],0x00,strlen(buffer));
        sprintf (&buffer[0], "CMD+CONTOAP=jaguar,shruthi@\r\n");
        HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
        //memset(&buffer[0],0x00,strlen(buffer));
        HAL_Delay(2000);
        HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
        HAL_Delay(500);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);

        /********* CMD?WIFI**********/
        //memset(&buffer[0],0x00,strlen(buffer));
        sprintf (&buffer[0], "CMD?WIFI\r\n");
        HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
       //memset(&buffer[0],0x00,strlen(buffer));
      //HAL_Delay(500);
```

```
	HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
	HAL_Delay(500);
	HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);

}
```

The code first declares a buffer of 128 characters. The buffer will be used to store the commands that are sent to the WE10 module.

The next few lines of code send the CMD+RESET command to the WE10 module. This command resets the module to its default state.

The next line of code sends the CMD+WIFIMODE=1 command to the WE10 module. This command sets the module to operate in WiFi mode.

The next line of code sends the CMD+CONTOAP=SSID, PASSWD command to the WE10 module. This command configures the module to connect to the WiFi network with the specified SSID and password.

The next line of code sends the CMD.WIFI command to the WE10 module. This command queries the module for its WiFi status.

The last line of code waits for 2000 milliseconds and then receives a response from the WE10 module. The response is stored in the buffer.

The WE10_Init() function is a simple example of how to initialize a WE10 module and connect it to a WiFi network. The function takes no arguments and it returns void.

**10.CODE SNIPPET TO MQTT INITIALIZATION AND CONNECTION WITH INTERRUPT-BASED UART COMMANDS**

```
void MQTT_Init()
{

	char buffer[128];

	/*********CMD+MQTTNETCFG **********/
	//memset(&buffer[0],0x00,strlen(buffer));
	sprintf (&buffer[0], "CMD+MQTTNETCFG=dev.rightech.io,1883\r\n");
	HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
	HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
	//memset(&buffer[0],0x00,strlen(buffer));
	//HAL_Delay(500);
	HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
	HAL_Delay(500);
	HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);


	/*********CMD+MQTTCONCFG---->LED **********/
	//memset(&buffer[0],0x00,strlen(buffer));
sprintf (&buffer[0], "CMD+MQTTCONCFG=3,mqtt-veereshpatil585228-wfg240,,,,,,,,,\r\n");
```

```
  HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
  HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
  //memset(&buffer[0],0x00,strlen(buffer));
  //HAL_Delay(500);
  HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
  HAL_Delay(500);
  HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);


  /*********CMD+MQTTSTART **********/
  //memset(&buffer[0],0x00,strlen(buffer));
  sprintf (&buffer[0], "CMD+MQTTSTART=1\r\n");
  HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
  HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
 //memset(&buffer[0],0x00,strlen(buffer));
  HAL_Delay(5000);
  HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
  HAL_Delay(500);
  HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);



  /*********CMD+MQTTSUB *********/
  //memset(&buffer[0],0x00,strlen(buffer));
  sprintf (&buffer[0], "CMD+MQTTSUB=base/relay/led1\r\n");
  HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
  HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
  HAL_Delay(500);
  HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
  HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);

}
```

The code you provided is a initialize a WE10 module and connect it to an MQTT broker. The code first declares a buffer of 128 characters. The buffer will be used to store the commands that are sent to the WE10 module.
The next few lines of code send the CMD+MQTTNETCFG command to the WE10 module. This command configures the module to connect to the MQTT broker at dev.rightech.io on port 1883. The CMD+MQTTCONCFG command configures the module to connect to the MQTT broker as a client with the username mqtt-arifm4348- ud8eo8 and no password. The CMD+MQTTSTART command starts the MQTT client and connects to the broker. The CMD+MQTTSUB command subscribes the client to the topic base/relay/led1.

The MQTT_Init() function is a simple example of how to initialize a WE10 module and connect it to an MQTT broker. The function takes no arguments and it returns void.
Here is a more detailed explanation of the code:
The CMD+MQTTNETCFG command is used to configure the MQTT parameters of the WE10 module. The first parameter is the hostname or IP address of the MQTT broker. The second parameter is the port number of the MQTT broker.
The CMD+MQTTCONCFG command is used to configure the MQTT client of the WE10 module. The first parameter is the username of the MQTT client. The second parameter is the password of the MQTT client.

The CMD+MQTTSTART command is used to start the MQTT client of the WE10 module. This command connects the client to the MQTT broker.
The CMD+MQTTSUB command is used to subscribe the MQTT client to a topic. The first parameter is the topic that the client wants to subscribe to.

**11.Send_Task Function**

```
void mqtt_data_send()
{
	char buffer[50];
	sprintf (&buffer[0], "CMD+MQTTPUB=base/state/value,%.2f\r\n",dis1);
	HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
	HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
	HAL_Delay(100);
}
```

This code is a function written in C that sends MQTT (Message Queuing Telemetry Transport) data using UART (Universal Asynchronous Receiver-Transmitter) communication.This declares a function named mqtt_data_send with no input parameters and no return value (void).
A character array (buffer) named buffer is declared with a size of 50 characters. This buffer will be used to store the formatted MQTT message.The sprintf function is used to format a string with the MQTT message. The message includes a command ("CMD+MQTTPUB"), a topic ("base/state/distance"), and a floating-point value (dis1) with two decimal places.
The formatted message stored in the buffer is transmitted via UART. The HAL_UART_Transmit function is used for this purpose. The message is sent to two UART interfaces (huart1 and huart2), and the third argument specifies the timeout duration (1000 milliseconds in this case).
A delay of 100 milliseconds is introduced using HAL_Delay. This delay allows time for the UART transmissions to complete before the function exits.In summary, this code snippet is part of a larger program, and its purpose is to format and transmit an MQTT message containing distance information (dis1) over two UART interfaces (huart1 and huart2). The delay at the end ensures that there is sufficient time for the transmissions to complete before moving on.

The code you provided defines the Send_Task function. The Send_Task is a task that will be executed by the code.
The Send_Task function first declares a variable of type data called DatatoSend.
The temp member of the data structure is used to store the temperature reading, and the humidity member of the data structure is used to store the humidity reading.
The Send_Task function then enters an infinite loop. In each iteration of the loop, the Send_Task function reads the temperature and humidity readings from the sensors, stores the readings in the DatatoSend structure, and then puts the DatatoSend structure on the myQueueTemp message queue.
The osMessageQueuePut() function is used to put a message on a message queue. The first parameter is the handle of the message queue, the second parameter is a pointer to the message, the third parameter is the priority of the message, and the fourth parameter is the timeout value.

**SendTask:**

The SendTask is a task that will be created by the code.The osThreadId_t SendTaskHandle variable is used to store the handle of the SendTask. The osThreadAttr_t SendTask_attributes structure defines the attributes of the SendTask.

The SendTask_attributes structure has three members:  name: The name of the task.stack_size The size of the stack that will be allocated to the task.  priority: The priority of the task.
In this case, the name of the task is "SendTask", the stack_size is 128 * 4 bytes, and the priority is osPriorityNormal.
The osThreadAttr_t structure is used to configure the attributes of a task. The name member is used to set the name of the task. The stack_size member is used to set the size of the stack that will be allocated to the task. The priority member is used to set the priority of the task.

**RecieveTask:**

The RecieveTask is a task that will be created by the code.
The osThreadId_t RecieveTaskHandle variable is used to store the handle of the RecieveTask. The osThreadAttr_t RecieveTask_attributes structure defines the attributes of the RecieveTask.
The RecieveTask_attributes structure has three members:  name: The name of the task.stack_size The size of the stack that will be allocated to the task.  priority: The priority of the task.
In this case, the name of the task is "RecieveTask", the stack_size is 128* 4 bytes, and the priority is osPriorityLow.
The osThreadAttr_t structure is used to configure the attributes of a task. The name member is used to set the name of the task. The stack_size member is used to set the size of the stack that will be allocated to the task. The priority member is used to set the priority of the task.

## 12.PROJECT CODE

## STAGE 2:(STM32F446RE WITH W10 MODULE)

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2023 STMicroelectronics.
  * All rights reserved.
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stdio.h"
#include "string.h"

uint8_t sensorStatus;

#define LedPin GPIO_PIN_5   // Onboard LED pin
#define LedPort GPIOA       // Onboard LED port

#define SensorPin GPIO_PIN_0
#define SensorPort GPIOA


/* Private variables ---------------------------------------------------------*/
```

```
UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART2_UART_Init(void);
void WE10_Init();
void MQTT_Init();

void mqtt_data_send(uint8_t n)
{
char buffer[50];

sprintf(&buffer[0], "CMD+MQTTPUB=reading/values,%d\r\n", n);
HAL_UART_Transmit(&huart1, (uint8_t *)buffer, strlen(buffer), 1000);
HAL_Delay(100);
}
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART1_UART_Init();
  MX_USART2_UART_Init();
  WE10_Init();
```

```
 MQTT_Init();
 /* USER CODE BEGIN 2 */

 /* USER CODE END 2 */

 /* Infinite loop */
 /* USER CODE BEGIN WHILE */

 char message[50];


          while (1)
          {

                  if (HAL_GPIO_ReadPin(SensorPort, SensorPin) == GPIO_PIN_RESET)

                    {
                          HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_RESET);


                  sprintf(message,"SENSOR NOT DETECTED ....\r\n");
                  HAL_UART_Transmit(&huart2, (uint8_t *)message, strlen(message), HAL_MAX_DELAY);
           HAL_Delay(500);
                  sprintf(message, "led status : %d\r\n", 0);
                  HAL_UART_Transmit(&huart2, (uint8_t *)message, strlen(message), HAL_MAX_DELAY);


                   mqtt_data_send(1);
                 }

        else
              {
                 // HAL_GPIO_TogglePin(Led_GPIO_Port, Led_Pin);

          HAL_GPIO_WritePin(GPIOA,GPIO_PIN_5,GPIO_PIN_SET);

                  sprintf(message,"SENSOR  DETECTED....\r\n");
                  HAL_UART_Transmit(&huart2, (uint8_t *)message, strlen(message), HAL_MAX_DELAY);
           HAL_Delay(500);
                  sprintf(message, "led status : %d\r\n", 1);
                  HAL_UART_Transmit(&huart2, (uint8_t *)message, strlen(message), HAL_MAX_DELAY);


                  mqtt_data_send(0);


               }
              HAL_Delay(500);

              // Optional delay to avoid rapid readings

            }
          }


void WE10_Init ()
{
          char buffer[128];
          /*** CMD+RESET ****/
          //memset(&buffer[0],0x00,strlen(buffer));
          sprintf (&buffer[0], "CMD+RESET\r\n");
          HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
          HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
```

```
	HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
	HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);


	/***  CMD+WIFIMODE=1  ****/
	//memset(&buffer[0],0x00,strlen(buffer));
	sprintf (&buffer[0], "CMD+WIFIMODE=1\r\n");
	HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
	HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);

	HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
	HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);


	/*** CMD+CONTOAP=SSID,PASSWD ****/
	//memset(&buffer[0],0x00,strlen(buffer));
	sprintf (&buffer[0],"CMD+CONTOAP=OnePlus Nord CE 3 Lite 5G,shruthi@\r\n");
	HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
	HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
	//memset(&buffer[0],0x00,strlen(buffer));
	HAL_Delay(2000);
	HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
	HAL_Delay(500);
	HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);

	/*** CMD?WIFI****/
	//memset(&buffer[0],0x00,strlen(buffer));
	sprintf (&buffer[0], "CMD?WIFI\r\n");
	HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
	HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
//	memset(&buffer[0],0x00,strlen(buffer));
//	HAL_Delay(500);
	HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
	HAL_Delay(500);
	HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);


}

void MQTT_Init()
{

	char buffer[128];

	/****CMD+MQTTNETCFG ***/
	//memset(&buffer[0],0x00,strlen(buffer));
	sprintf (&buffer[0], "CMD+MQTTNETCFG=dev.rightech.io,1883\r\n");
	HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
	HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);
	//memset(&buffer[0],0x00,strlen(buffer));
	//HAL_Delay(500);
	HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 10000);
	HAL_Delay(500);
	HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 10000);


	/****CMD+MQTTCONCFG---->LED ***/
	//memset(&buffer[0],0x00,strlen(buffer));
	sprintf (&buffer[0], "CMD+MQTTCONCFG=3,mqtt-veereshpatil585228-wfg240,,,,,,,,\r\n");
	HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
	HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
	//memset(&buffer[0],0x00,strlen(buffer));
```

```
        //HAL_Delay(500);
        HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
        HAL_Delay(500);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);


        /****CMD+MQTTSTART ***/
        //memset(&buffer[0],0x00,strlen(buffer));
        sprintf (&buffer[0], "CMD+MQTTSTART=1\r\n");
        HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
//      memset(&buffer[0],0x00,strlen(buffer));
        HAL_Delay(5000);
        HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
        HAL_Delay(500);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);

        /****CMD+MQTTSUB ****/
        //memset(&buffer[0],0x00,strlen(buffer));
        sprintf (&buffer[0], "CMD+MQTTSUB=base/relay/led1\r\n");
        HAL_UART_Transmit(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);
        HAL_Delay(500);
        HAL_UART_Receive(&huart1, (uint8_t*)buffer, strlen(buffer), 1000);
        HAL_UART_Transmit(&huart2, (uint8_t*)buffer, strlen(buffer), 1000);

}



void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
```

```
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{


  huart1.Instance = USART1;
  huart1.Init.BaudRate = 38400;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}


static void MX_USART2_UART_Init(void)
{


  huart2.Instance = USART2;
  huart2.Init.BaudRate = 38400;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}



static void MX_GPIO_Init(void)
```

```
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();



  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pin : PA0 */
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : PA5 */
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}


void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

**STAGE:3**

```
#include<stdio.h>
#include<string.h>
#include<errno.h>
#include<stdlib.h>

void error(const char *msg)
{
   perror(msg);
   exit(1);
}

int main()
{
   const char *portname = "/dev/ttyS3";  // Replace with the actual UART device file

   int fd = open(portname, O_RDWR | O_NOCTTY | O_SYNC);
   if (fd < 0)
{
      error("Error opening UART");
   }

   struct termios tty;
   if (tcgetattr(fd, &tty) < 0)
 {
      error("Error from tcgetattr");
   }

   cfsetospeed(&tty, B38400);  // Set the baud rate
   cfsetispeed(&tty, B38400);

   tty.c_cflag |= (CLOCAL | CREAD);    // Ignore modem control lines, enable receiver
   tty.c_cflag &= ~CSIZE;            // Clear data size bits
   tty.c_cflag |= CS8;              // 8-bit data
   tty.c_cflag &= ~PARENB;           // No parity bit
   tty.c_cflag &= ~CSTOPB;           // 1 stop bit
   tty.c_cflag &= ~CRTSCTS;          // No hardware flow control

   tty.c_lflag = 0;                // Non-canonical mode

   tty.c_cc[VMIN] = 1;              // Minimum number of characters to read
   tty.c_cc[VTIME] = 1;             // Time to wait for data (in tenths of a second)

   if (tcsetattr(fd, TCSANOW, &tty) != 0)
{
      error("Error from tcsetattr");
   }
while(1)
{

   char buf[50];
   memset(buf, 0, sizeof(buf));
```

```
    int n = read(fd, buf, sizeof(buf));
    if (n < 0) {
        error("Error reading");
    }

    printf("Received: %s\n", buf);
}
    close(fd);
    return 0;
}
```

**STAGE:4 (RUGGED BOARD WITH W10)**

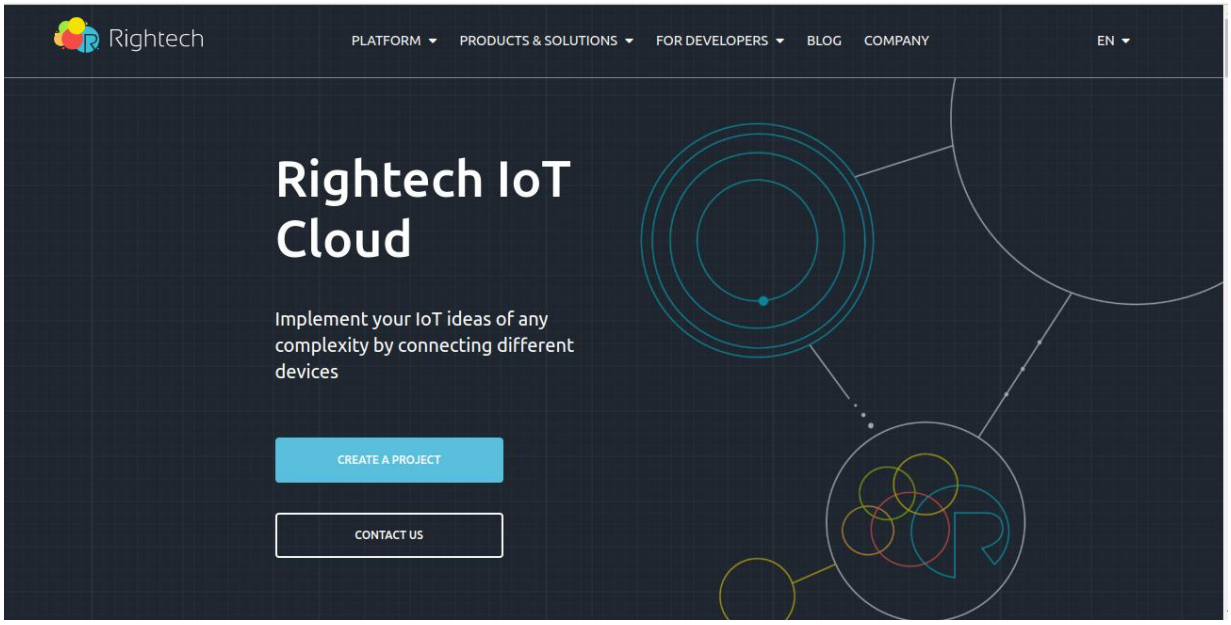
```
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <termios.h>
#include <unistd.h>
int set_interface_attribs(int fd, int speed)
{
        struct termios tty;

        if (tcgetattr(fd, &tty) < 0)
        {
                printf("Error from tcgetattr: %s\n", strerror(errno));
                return -1;
        }

        cfsetispeed(&tty, (speed_t)speed);
        tty.c_cflag |= (CLOCAL | CREAD);    /* ignore modem controls */
        tty.c_cflag &= ~CSIZE;
        tty.c_cflag |= CS8;         /* 8-bit characters */
        tty.c_cflag &= ~PARENB;     /* no parity bit */
        tty.c_cflag &= ~CSTOPB;     /* only need 1 stop bit */
        tty.c_cflag &= ~CRTSCTS;    /* no hardware flowcontrol */


        tty.c_iflag = IGNPAR;
        tty.c_lflag = 0;


        tty.c_cc[VMIN] = 1;
        tty.c_cc[VTIME] = 1;

        if (tcsetattr(fd, TCSANOW, &tty) != 0)
        {
                printf("Error from tcsetattr: %s\n", strerror(errno));
                return -1;
        }
        return 0;
```

```
}

int main()
{
    char *portname = "/dev/ttyS3";
    int fd;
    int wlen;
    int rdlen;
    int ret;

    char res[5];
    char arr1[] = "CMD+RESET\r\n";
    char arr2[] = "CMD+WIFIMODE=1\r\n";
    char arr[] = "CMD+CONTOAP=\"OnePlus Nord CE 3 Lite 5G\",\"shruthi@\"\r\n";
    char arr3[] = "CMD+MQTTNETCFG=dev.rightech.io,1883\r\n";
    char arr4[] = "CMD+MQTTCONCFG=3,mqtt-veereshpatil585228-wfg240,,,,,,,,,\r\n";
    char arr5[] = "CMD+MQTTSTART=1\r\n";
    char arr6[] = "CMD+MQTTSUB=base/relay/led1\r\n";

    unsigned char buf[100];

    fd = open(portname, O_RDWR | O_NOCTTY | O_SYNC);
    if (fd < 0)
    {
        printf("Error opening %s: %s\n", portname, strerror(errno));
        return -1;
    }
    set_interface_attribs(fd, B38400);

    printf("%s", arr1);
    wlen = write(fd, arr1, sizeof(arr1) - 1);
    sleep(3);


    // Send CMD+WIFIMODE=1
    printf("%s", arr2);
    wlen = write(fd, arr2, sizeof(arr2) - 1);
    sleep(3);


    // Send CMD+CONTOAP
    printf("%s", arr);
    wlen = write(fd, arr, sizeof(arr) - 1);
    sleep(3);


    printf("%s", arr3);
    wlen = write(fd, arr3, sizeof(arr3) - 1);
    sleep(3);


    printf("%s", arr4);
```

```
    wlen = write(fd, arr4, sizeof(arr4) - 1);
    sleep(3);



    printf("%s", arr5);
    wlen = write(fd, arr5, sizeof(arr5) - 1);
    sleep(3);

    printf("%s", arr6);
    wlen = write(fd, arr6, sizeof(arr6) - 1);
    sleep(3);

char buffer[100]; // Create a buffer to hold the formatted message

while(1){
rdlen = read(fd, buf, sizeof(buf) - 1);
if (rdlen > 0) {
    buf[rdlen] = '\0'; // Null-terminate the received data
        printf("%s\n", buf);

    int ret = snprintf(buffer, sizeof(buffer), "CMD+MQTTPUB=led status,%s\r\n", buf);

    if (ret < 0) {

    }
else
{

        ssize_t wlen = write(fd, buffer, ret);
        //sleep(3);
        if (wlen == -1) {

        }
    }
}
}

    close(fd);
    return 0;
}
```

**13.RIGHT TECH OUTPUT**

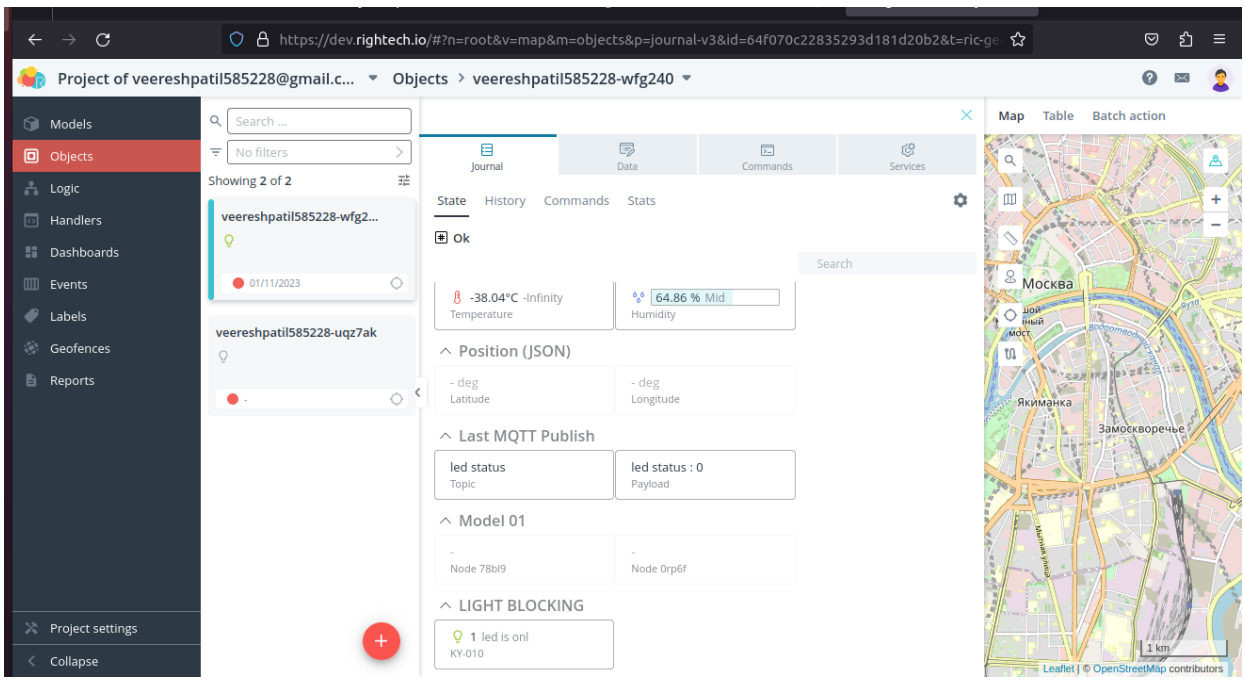
Rightech IoT Cloud is a tool for developers. RIC is independent of specific equipment and protocols, which makes it easier for developers to combine different devices under one solution.Platform tools allow developers to create IoT solutions without extra code and reuse 90% of that solution to launch similar cases.




We can download the stored data from Rightech IoT in different formats and those are GPX, or GPS Exchange Format, is an XML schema designed as a common GPS data format for software applications.

CSV (comma-separated values) file is a text file that has a specific format that allows data to be saved in a table-structured format.

JSON (JavaScript Object Notation, pronounced /ˈdʒeɪsən/; also /ˈdʒeɪˌsɒn/) is an open standard file format and data interchange formatthat uses human-readable text to store and transmit data objects consisting of attribute–value pairs and arrays (or other serializable values).I have downloaded it in JSON format which is given below.



## 14. REAL TIME APPLICATION

The KY-010 Photo Interrupter Module, with its ability to detect interruptions in an infrared light beam, finds applications in various real-time scenarios.

1. **Object Detection:**
    - The primary function of the KY-010 module is object detection. It can be used in applications where the presence or absence of an object needs to be monitored in real-time. For example, it could be employed in conveyor systems to detect the passage of items.
2. **Line Following Robots:**
    - Line following robots use infrared sensors to detect lines on a surface. The KY-010 module can be integrated into such robots to follow a predefined path. When the sensor detects a line, the robot adjusts its direction accordingly.
3. **Rotary Encoder Systems:**
    - The KY-010 module can be part of rotary encoder systems where it senses the rotation of a wheel or disk. This is useful in applications like position tracking or speed measurement, commonly found in robotics or motor control systems.
4. **Automated Gate Systems:**
    - In security or access control systems, the KY-010 module can be utilized to detect the presence of a vehicle or person, triggering the opening or closing of gates in real-time.
5. **Paper or Object Counting:**
    - In industries or settings where counting objects or pieces of paper is crucial, the KY-

010 module can be employed to count each instance as an object interrupts the infrared beam.

6. **Interactive Displays:**
    - The module can be used in interactive displays where the interruption of the infrared beam by a user's hand or an object triggers a specific response or action on the display.
7. **Photocopier or Printer Paper Feed Detection:**
    - KY-010 modules can be integrated into photocopiers or printers to detect the presence of paper in the feed tray. This ensures proper paper handling during printing or copying processes.
8. **Coin or Token Detection:**
    - In vending machines or coin-operated devices, the KY-010 module can be used to detect the presence of coins or tokens as they pass through a specific point in the machine.
9. **Security Systems:**
    - The module can be a component of security systems, triggering alarms or alerts when an object or person is detected in a restricted area.
10. **Sorting Systems:**
    - In sorting applications, such as those found in logistics or manufacturing, the KY-010 module can assist in the identification and sorting of objects based on their presence or absence.

## 15. CONCLUSION

- **Working Principle**: The KY-010 module works based on the interruption of the infrared light beam. When an object comes between the IR LED and the photodiode, it blocks the light, causing the photodiode to detect a decrease in light intensity.
- **Output Signal:** The module typically provides a digital output signal, indicating whether the light beam is interrupted or not. This makes it easy to interface with microcontrollers and digital systems.
- **Applications:** KY-010 photo interrupter modules are commonly used in various applications such as object detection, counting systems, speed measurement, and position sensing.
- **Usage with Arduino and Other Microcontrollers:** These modules are often used in conjunction with microcontrollers like Arduino for projects involving object detection or automation.
- **Adjustability:** Some KY-010 modules come with adjustable components to set sensitivity or threshold levels, allowing users to customize the module for specific applications.

- **Limitations:** The accuracy and reliability of the module may be affected by ambient light conditions, and adjustments may be required for optimal performance in different environments.
- **Documentation:** It's crucial to refer to the module's datasheet or documentation for specific details on pin configuration, electrical characteristics, and usage guidelines.
- **Consideration of Distance:** The effective range between the IR LED and photodiode should be considered, and the module may have limitations on the maximum and minimum distances for reliable operation.


**IF ANY QUERYS CONTACT ME:**

**NAME: VEERESH PATIL**

**CONTACT: 7829267565**

**EMAIL: veereshpatil585228@gmail.com**