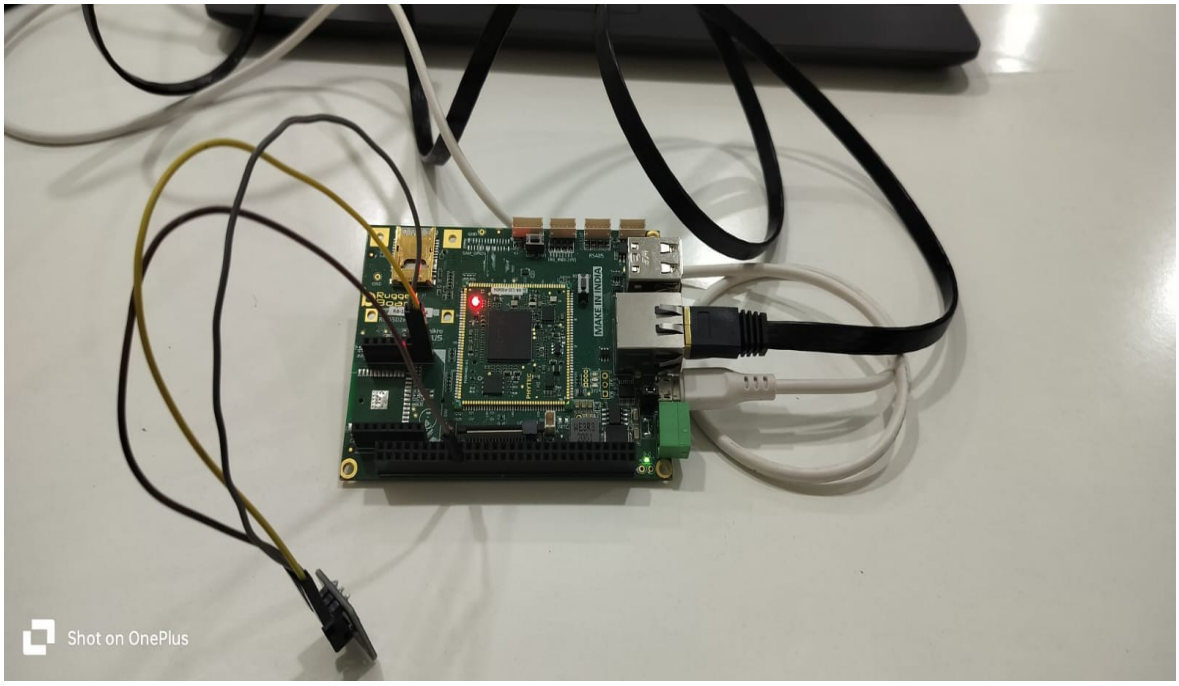
KY-010 PHOTO INTERRUPT MODULE with RUGGEDBOARD-a5d2x Project



PRESENTED BY
VEERESH P
DECEMBER-2023

TABLE OF CONTENT

1.Project Summary

The provided C code appears to be a part of a project that involves interfacing with ky-010 Photo interrupt sensor. The code is designed to run on a system with the MRAA library, which provides abstractions for interacting with GPIO pins on various platforms.

2.Key Features of the Project:

PHOTO INTERRUPT MODULE:

The project interfaces with ky-010 sensors using GPIO pins (GPIO_LED and GPIO_SIGNAL) to turn on the led when object is detected based on the interructions.

MRAA Library Usage:

The MRAA library is utilized for GPIO operations, providing a higher-level interface for working with GPIO pins compared to direct sysfs manipulation.

GPIO Initialization and Direction Setting:

The setup function initializes the GPIO pins (echo_pin and signal_pin) using the MRAA library. It sets the direction of the pins, designating echo_pin as an input and signal_pin as an output.
Trigger Pulse Generation:

The triggerPulse function generates a pulse on the trigger pin to initiate the ultrasonic measurement. It writes a logical high (1) for a brief period and then sets it back to low (0).
Distance Measurement Loop:

The loop function runs in an infinite loop, triggering ultrasonic pulses and measuring the time it takes for the echo to return.
It utilizes the gettimeofday function to capture the start and end times of the echo.

The distance is calculated based on the elapsed time and the speed of sound (343 meters/second).
The results are printed to the console, including the elapsed time and the calculated distance in centimeters.
Cleanup:

The main function calls the setup function to initialize GPIO pins and then enters the main loop using the loop function.
Proper cleanup is implemented, closing the GPIO pins (echo_pin and trigger_pin) using mraa_gpio_close before exiting the program.
Platform Portability:

The use of the MRAA library enhances platform portability, allowing the code to be easily adapted to different hardware configurations that are supported by MRAA.
It's important to note that the success of the project depends on the correctness of GPIO pin assignments, hardware compatibility, and the accuracy of distance calculations. Additionally, proper error handling and edge case considerations should be implemented for a robust and reliable system.

3.Hardware used and there Specification

Rugged Board - A5D2x is an Single Board Computer providing as easy migration path from Micro controller to Microprocessor. Rugged Board 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.

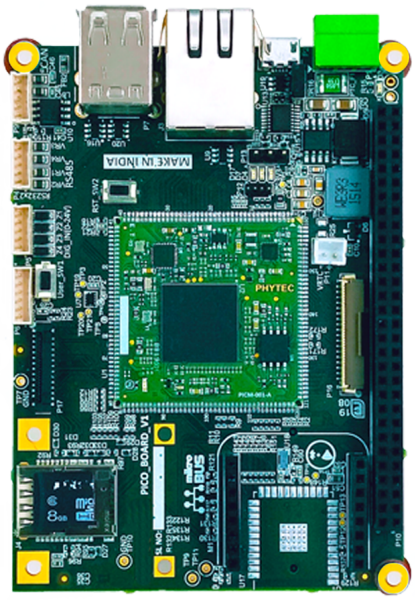
RuggedBoard - A5D2x Specification:

System On Module
SOC Microchip ATSAMA5d2x Cortex-A5
Frequency 500MHz
RAM 64 MB DDR3
Flash32 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



A5D2x @500MHz
Cortex - A5
64MB Ram
32MB Flash

2 x USB

DC & USB Power

2 x RS232

Expansion Header

1x RS485

mikroBUS Conn.

1 x CAN

1 x Ethernet

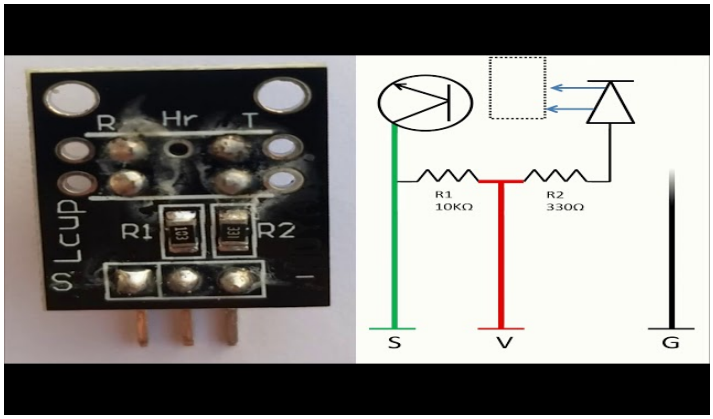
mPCIe conn.

TFT & Cap Touch

Micro Sim Slot

1 x MicroSD Slot

KY-010 MODULE



Power Supply :+5V DC
Quiescent Current : <2mA
Working Current: 15mA
Effectual Angle: <15°
Ranging Distance : 2cm – 400 cm/1" -13ft
Resolution : 0.3 cm
Measuring Angle: 30 degree
Trigger Input Pulse width: 10uS
Dimension: 45mm x 20mm x 15mm

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.

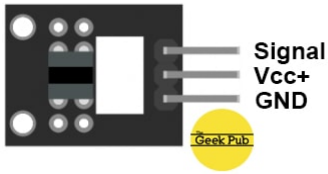
OPERATION:

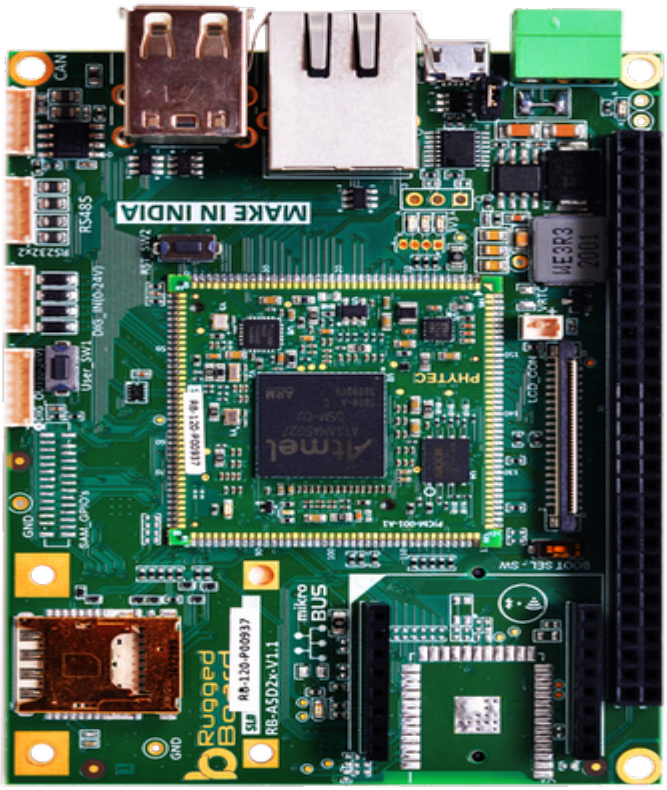
The KY-010 Photo Interrupter Module operates based on the interruption of an infrared (IR) light beam.

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.Connection diagram





PIN CONNECTIONS:

PIN	PIN NUMBER	COMPONENTS
VCC 5V	VCC	KY-010 (+5V)

GROUND	GND	KY-010 (GND)
PC13	61 (MRAA)	LED pin
PA31	41 (MRAA)	SIGNAL pin

5.Project Code and explanation

a)sys_main code:

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

#define DIGITAL_PIN "/sys/class/gpio/PA31/value"  // Replace with the actual GPIO pin for the KY-010 digital output
#define LED_PIN "/sys/class/gpio/PC13/value"  // Replace with the actual GPIO pin for the LED

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

int readDigitalValue() {
    FILE *fp = fopen(DIGITAL_PIN, "r");
    if (fp == NULL) {
        error("Error opening digital pin");
    }

    char value;
```

```
    fscanf(fp, " %c", &value);  // Leading space to skip any leading whitespace
    fclose(fp);

    // '1' indicates the presence of a signal, '0' indicates no signal
    return (value == '1');
}

void controlLED(int state) {
    FILE *ledFile = fopen(LED_PIN, "w");
    if (ledFile == NULL) {
        error("Error opening LED pin");
    }

    fprintf(ledFile, "%d", state);
    fclose(ledFile);
}

int main() {
    // Assuming GPIO pin 17 is not exported, you might need to export it first
    FILE *exportFile = fopen("/sys/class/gpio/export", "w");
    if (exportFile == NULL) {
        error("Error exporting GPIO pin");
    }

    fprintf(exportFile, "31");  // KY-010 digital output GPIO pin
    fclose(exportFile);

    // Set the GPIO pin direction to in (for reading the KY-010 digital output)
    FILE *sensorDirectionFile = fopen("/sys/class/gpio/PA31/direction", "w");
    if (sensorDirectionFile == NULL) {
        error("Error setting direction for KY-010 digital output pin");
    }

    fprintf(sensorDirectionFile, "in");
    fclose(sensorDirectionFile);
```

```
    // Assuming GPIO pin 23 is not exported, you might need to export it
first
    FILE *exportLEDFile = fopen("/sys/class/gpio/export", "w");
    if (exportLEDFile == NULL) {
        error("Error exporting LED pin");
    }

    fprintf(exportLEDFile, "13");  // LED GPIO pin
    fclose(exportLEDFile);

    // Set the GPIO pin direction to out (for controlling the LED)
    FILE *ledDirectionFile = fopen("/sys/class/gpio/PC13/direction", "w");
    if (ledDirectionFile == NULL) {
        error("Error setting direction for LED pin");
    }

    fprintf(ledDirectionFile, "out");
    fclose(ledDirectionFile);

    while (1) {
        int photoInterruptState = readDigitalValue();

        if (photoInterruptState) {
            printf("Object detected - LED on\n");

            controlLED(0);  // Turn on the LED
        } else {
            printf("no object detected - LED off\n");
            controlLED(1);  // Turn off the LED
        }

        // Adjust the sleep duration based on your sensor's update rate
        sleep(1);  // Sleep for 1 second between readings
    }

    // Unexport GPIO pins before exiting
    FILE *unexportFile1 = fopen("/sys/class/gpio/unexport", "w");
    if (unexportFile1 == NULL) {
```

```
        error("Error unexporting KY-010 digital output pin");
    }

    fprintf(unexportFile1, "31");
    fclose(unexportFile1);

    FILE *unexportFile2 = fopen("/sys/class/gpio/unexport", "w");
    if (unexportFile2 == NULL) {
        error("Error unexporting LED pin");
    }

    fprintf(unexportFile2, "13");
    fclose(unexportFile2);

    return 0;
}
```

In the main function, it calls the setup function to initialize GPIO pins and then enters the main loop using the loop function. The cleanup section is mentioned but not implemented in this example. It's important to close file descriptors and unexport GPIO pins properly to avoid issues. You would typically implement a cleanup function to handle these tasks.

b.mraa code

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <mraa/gpio.h>

#define DIGITAL_PIN 41   // Replace with the actual GPIO pin for the KY-010 digital output
#define LED_PIN 61       // Replace with the actual GPIO pin for the LED

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

mraa_gpio_context digitalPin, ledPin;
```

```
void setup() {
    // Export and set direction for KY-010 digital output pin
    digitalPin = mraa_gpio_init(DIGITAL_PIN);
    if (digitalPin == NULL) {
        error("Error initializing KY-010 digital output pin");
    }
    mraa_gpio_dir(digitalPin, MRAA_GPIO_IN);

    // Export and set direction for LED pin
    ledPin = mraa_gpio_init(LED_PIN);
    if (ledPin == NULL) {
        error("Error initializing LED pin");
    }
    mraa_gpio_dir(ledPin, MRAA_GPIO_OUT);
}

int readDigitalValue() {
    return mraa_gpio_read(digitalPin);
}

void controlLED(int state) {
    mraa_gpio_write(ledPin, state);
}

void cleanup() {
    mraa_gpio_close(digitalPin);
    mraa_gpio_close(ledPin);
}

int main() {
    setup();

    while (1) {
        int photoInterruptState = readDigitalValue();

        if (photoInterruptState) {
            printf("Object detected - LED on\n");
            controlLED(0);  // Turn on the LED
```

```
        } else {
            printf("No object detected - LED off\n");
            controlLED(1);  // Turn off the LED
        }

        // Adjust the sleep duration based on your sensor's update rate
        sleep(1);  // Sleep for 1 second between readings
    }

    cleanup();

    return 0;    }
```

The loop function contains the main loop of the program. It triggers a pulse using the triggerPulse function, waits for the echo response by polling the Echo pin, measures the time taken, calculates the distance based on the speed of sound, and then prints the elapsed time and calculated distance. It introduces a delay of 500,000 microseconds (0.5 seconds) for stable readings.

```
int main() {
    // Setup GPIO
    setup();

    // Run the main loop
    loop();

    // Cleanup
    mraa_gpio_close(LED_pin);
    mraa_gpio_close(SIGNAL_pin);

    return 0;
}
```

In the main function, it calls the setup function to initialize the GPIO pins, then enters the main loop using the loop function. After the main loop, it performs cleanup by closing the Echo and Trigger GPIO pins using mraa_gpio_close.