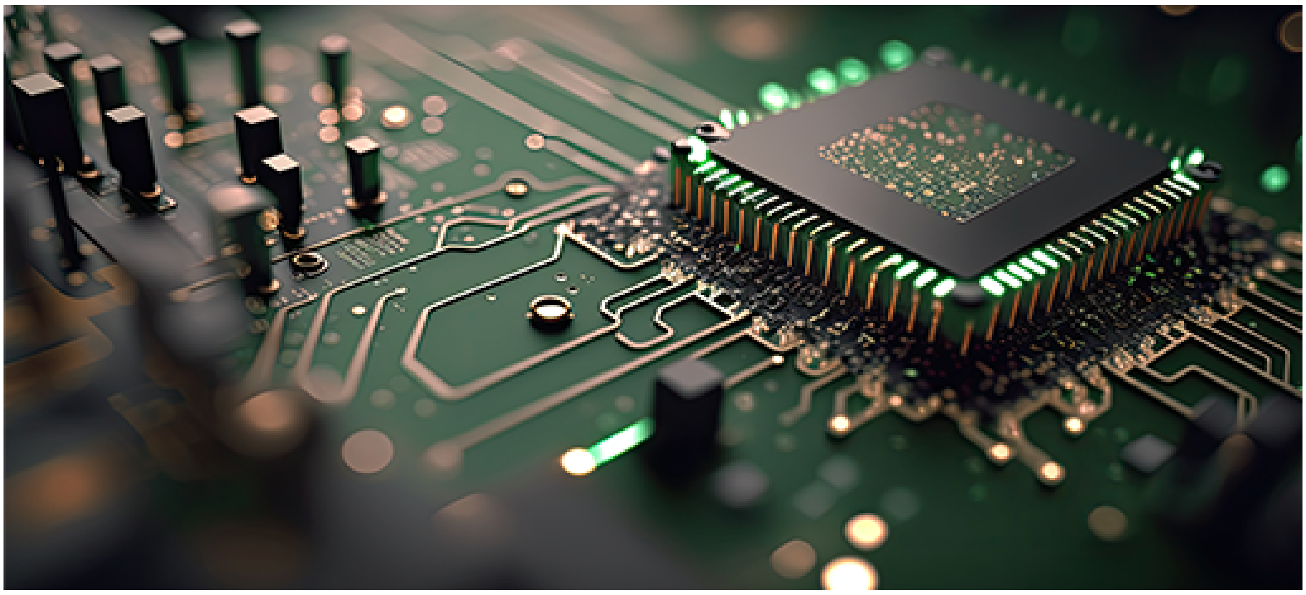
KY-020 BALL SWITCH SENSOR INTERFACING WITH RUGGUDBOARD-A5D2X



Presented by:

Manjunath E L
6366035611

TABLE OF CONTENTS

1. INTRODUCTION

The KY-020 Ball Switch Sensor is a module designed for detecting the tilt or movement of a device. It consists of a small metal ball inside a cylindrical container, and when the sensor is tilted, the ball moves, causing the module to change its electrical state. The module is commonly used in projects that require tilt or movement detection, such as robotics, gaming controllers, or orientation sensing applications.

Interfacing the KY-020 Ball Switch Sensor with the RugGudBoard-A5D2X involves connecting the sensor to the microcontroller and utilizing its input/output pins to read the sensor's state. The RugGudBoard-A5D2X, presumably a microcontroller board, could be programmed to respond to the sensor's output and trigger specific actions based on the tilt or movement detected.

Objective:

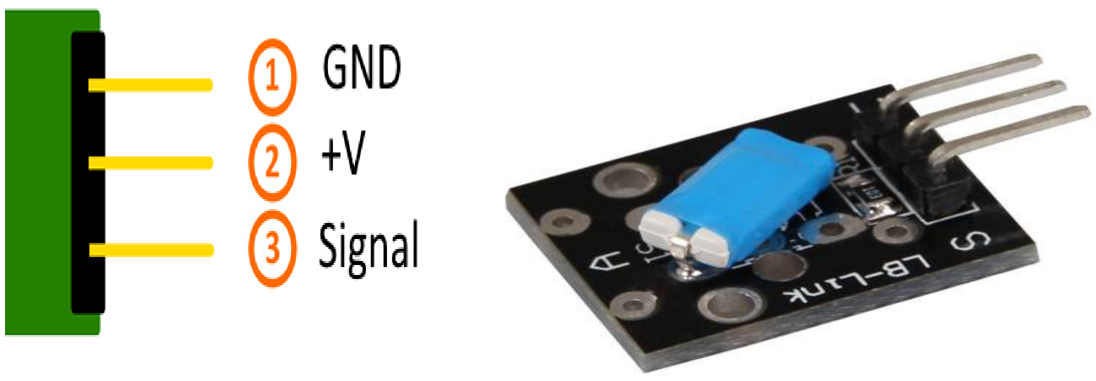
Create a tilt or movement detection system by integrating the KY-020 Ball Switch Sensor with the RugGudBoard-A5D2X, enabling the microcontroller to respond to changes in the sensor's state. This project may find applications in areas such as robotics, gaming controllers, or other systems where tilt or movement needs to be detected and acted upon.

2. Hardware Used and there Specification

1. Ball Switch Sensor:

Specification:

- ❖ Type: Normally Open (NO) Ball Switch
- ❖ Operating Voltage: 3V - 5V
- ❖ High Sensitivity
- ❖ Digital Outputs
- ❖ 3 Pin sensor Module
- ❖ Economic,Reliable and small in size



Ball switch sensor pin configuration

2. Ruggudboard

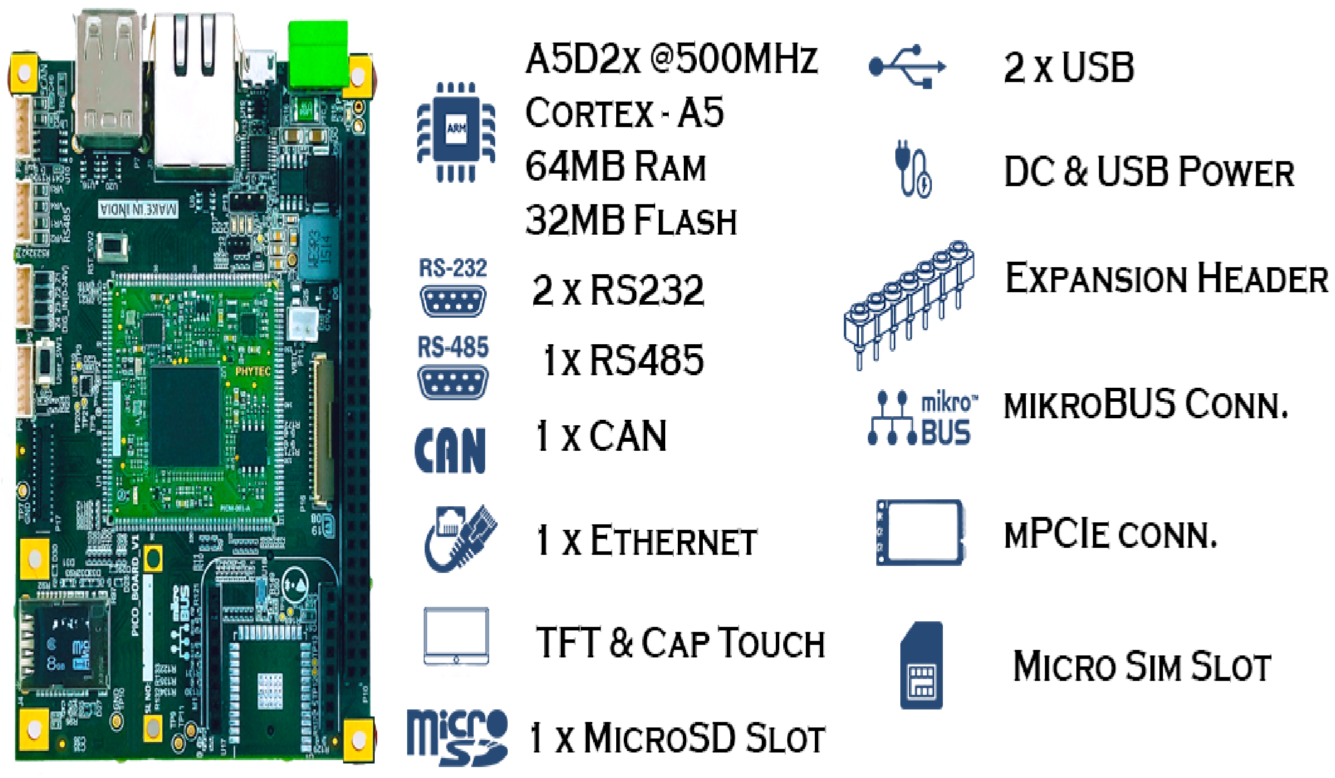
RuggedBoard - A5D2x is an Single Board Computer providing as easy migration path from Microcontroller to Microprocessor. RuggedBoard is enabled with industry Standard Yocto Build Embedded Linux platform and open source libraries for industrial application development.

Highlights

● Off-the-shelf Single Board Computer for IIoT

● Feature rich & Highly cost optimized platform for industrial Usage.

● Simple & Powerful Open Source Software IoT Stack



❖ The Open Source IoT stack equivalent to Intel MRAA & UPM developed in C/C++ holds a tremendous opportunity to the Embedded Systems & gives programmers its ease of Python / nodeJs / Java.

❖ Rugged Board team targets to combine the Open source (Carrier Boards) community strength with industrial grade SOM and initiated the first Open Source Hardware "Industrial Pico Computer" which is powered by phyCORE-A5D2x SOM with Microchip A5D2x Cortex-A5 Core @500 MHz.

3. SOURCE CODES OF THE PROJECT

Using the GPIO Pins:

```
#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 ball switch

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 (tilted), '0' indicates no signal (not
tilted)
    return (value == '1');
}

int main() {
    // Assuming GPIO pin 31 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");
    fclose(exportFile);

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

    fprintf(directionFile, "in");
    fclose(directionFile);

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

        if (ballSwitchState) {
            printf("Not Tilted=0\n");
        } else {
            printf("Tilted=1\n");
        }

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

    // Unexport GPIO pin 31 before exiting
    FILE *unexportFile = fopen("/sys/class/gpio/unexport", "w");
    if (unexportFile == NULL) {
        error("Error unexporting GPIO pin");
    }

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

    return 0;
}
```

Pin configuration of above code:

The code you provided is written in C and interacts with a GPIO (General Purpose Input/Output) pin on a Linux system. Here's a brief explanation of the pin configuration and how the code works:

1. **Digital Pin Configuration:**
   - The digital pin is specified by the macro DIGITAL_PIN and set to "/sys/class/gpio/PA31/value." This path corresponds to a GPIO pin

on the PA31 port. You might need to replace it with the actual GPIO pin you're using for the ball switch.

2. **Exporting GPIO Pin:**
    - The code attempts to export the GPIO pin by writing the pin number ("31") to the "/sys/class/gpio/export" file. This is necessary to make the GPIO pin accessible in user space.
3. **Setting Direction:**
    - The direction of the GPIO pin is set to "in" by writing "in" to the "/sys/class/gpio/PA31/direction" file. This configures the pin for input, as it reads the state of the ball switch.
4. **Reading Digital Value:**
    - The readDigitalValue function opens the "/sys/class/gpio/PA31/value" file, reads the digital value (either '0' or '1') indicating the state of the ball switch (tilted or not tilted), and returns it.
5. **Main Loop:**
    - In the main loop, the code continuously reads the state of the ball switch using the readDigitalValue function.
    - If the ball switch is tilted, it prints "Tilted=1"; otherwise, it prints "Not Tilted=0."
    - The loop sleeps for 1 second between readings (sleep(1)), but you may need to adjust this duration based on the sensor's update rate.
6. **Unexporting GPIO Pin:**
    - Before exiting the program, the GPIO pin is unexported by writing the pin number ("31") to the "/sys/class/gpio/unexport" file.

Using MRAA Library code:

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <mraa.h>


#define GPIO_PIN 36  // Replace with the actual GPIO pin for the ball switch


void error(const char *msg) {

    fprintf(stderr, "%s\n", msg);

    exit(1);

}


int readDigitalValue(mraa_gpio_context gpio) {

    return mraa_gpio_read(gpio);

}


int main() {

    mraa_result_t status = MRAA_SUCCESS;

    mraa_gpio_context gpio;


    // Initialize MRAA

    mraa_init();


    // Initialize GPIO pin

    gpio = mraa_gpio_init(GPIO_PIN);

    if (gpio == NULL) {

        error("Error initializing GPIO pin");
    }

    // Set the GPIO pin direction to input
    status = mraa_gpio_dir(gpio, MRAA_GPIO_IN);
    if (status != MRAA_SUCCESS) {
        error("Error setting direction for GPIO pin");
    }
    while (1) {
        int ballSwitchState = readDigitalValue(gpio);

        if (ballSwitchState) {
            printf("Not Tilted=0\n");
        } else {
            printf("Tilted=1\n");
        }

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

    // Close and cleanup GPIO pin before exiting
    mraa_gpio_close(gpio);
    return 0;
}
```

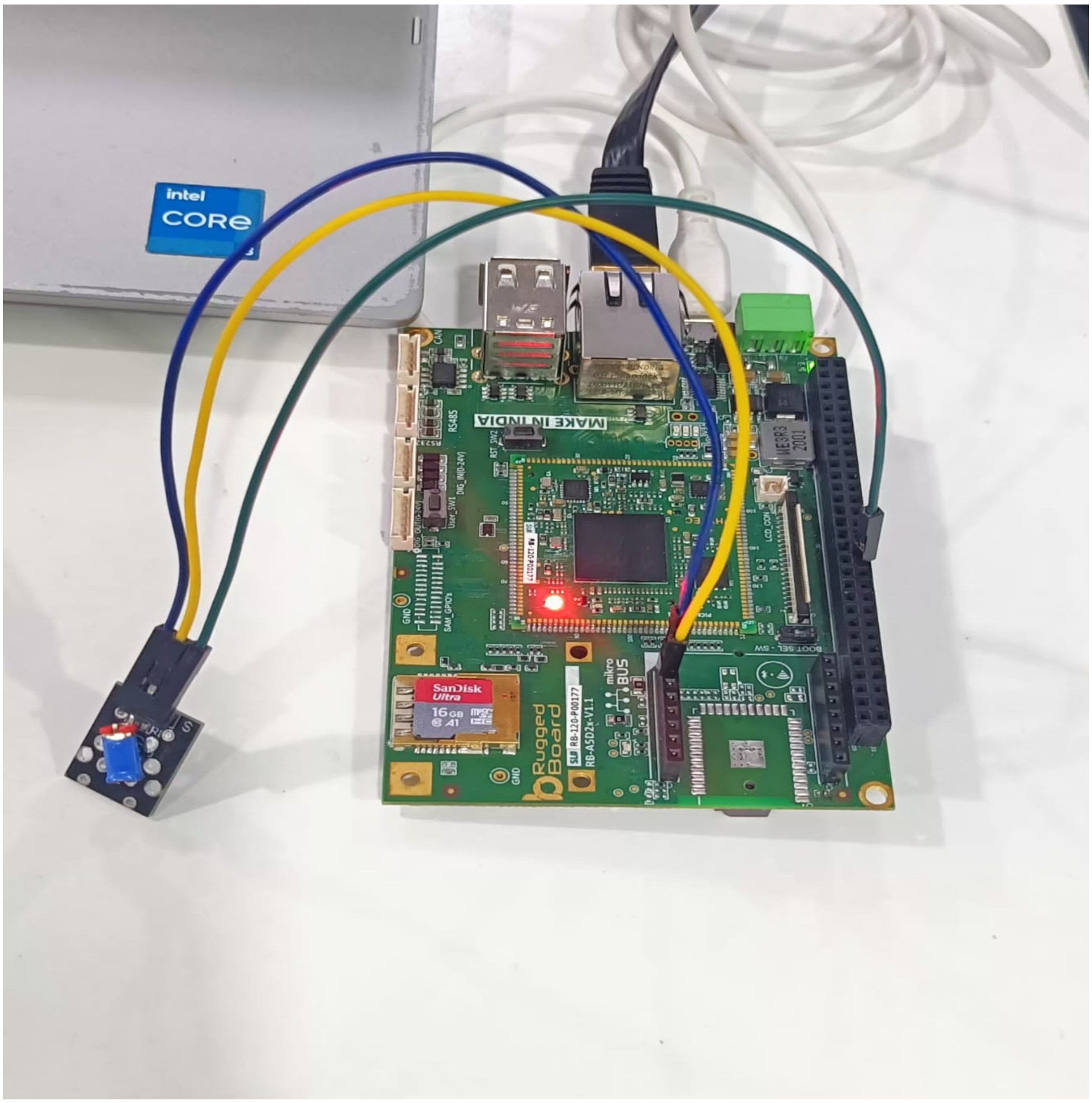
Pin Configuration of above code:

Here's an explanation of the pin configuration in the code:

1. **Include MRAA Library:**#include <mraa.h>: This line includes the MRAA library, which is a low-level library for interacting with the I/O on various platforms.
2. **Define GPIO_PIN:**
   - #define GPIO_PIN 31: This macro defines the GPIO pin number used for the ball switch. You should replace it with the actual GPIO pin you are using.
3. **Initialize MRAA and GPIO:**
   - mraa_init(): Initializes the MRAA library.
   - gpio = mraa_gpio_init(GPIO_PIN): Initializes the GPIO pin using the specified GPIO_PIN. The mraa_gpio_init function returns a context (gpio) that is used for subsequent operations.
4. **Set GPIO Direction:**
   - status = mraa_gpio_dir(gpio, MRAA_GPIO_IN): Sets the direction of the GPIO pin to input. The MRAA_GPIO_IN constant indicates input direction. The status variable checks if the operation was successful.
5. **Read Digital Value:**
   - readDigitalValue(gpio): Calls the readDigitalValue function to read the digital value of the GPIO pin using mraa_gpio_read(gpio).
6. **Main Loop:**
   - The while (1) loop continuously reads the state of the ball switch using the readDigitalValue function.
   - If the ball switch is tilted (ballSwitchState is non-zero), it prints "Not Tilted=0"; otherwise, it prints "Tilted=1."

    - The loop sleeps for 1 second between readings (sleep(1)), but you may need to adjust this duration based on the sensor's update rate.
7. **Cleanup GPIO Pin:**
    - mraa_gpio_close(gpio): Closes and cleans up the GPIO pin before exiting the program.

4. CONNECTION DIAGRAM OF THE PROJECT



Connection diagram of the project

5. CONCLUSION

1. **Linux Approach:**
   - The first code snippet relies on accessing GPIO pins through the Linux file system. It uses standard file I/O operations to read the state of the ball switch.
   - It provides a basic example of interfacing with GPIO pins, exporting, setting direction, and reading values.
   - The GPIO pin configuration in the code is specific to the Linux system and may need adjustment based on the actual GPIO pin used.
2. **MRAA Library Approach:**
   - The second code snippet utilizes the MRAA library, which abstracts low-level I/O operations, providing a more convenient interface for Intel Galileo or Edison boards.
   - It initializes the MRAA library, sets up the GPIO pin using MRAA functions, and reads the digital value with MRAA GPIO functions.
   - The MRAA approach simplifies the code and makes it more portable across different platforms, especially for Intel-based boards.

Considerations for Future Development:

- Sensor Calibration: Depending on the specific characteristics of the ball switch sensor, calibration may be required for accurate tilt detection.
- Integration with Other Sensors: The project can be extended by integrating additional sensors or actuators to create more complex applications.

THANK YOU