

```
fn main() {  
    let WHO_AM_I: &str = "Vaishnav Sabari Girish";  
  
    let designation: &str = "Final Year ECE Student";  
  
    let mut interests: Vec<&str> = vec![  
        "Open Source",  
        "Embedded Systems",  
        "Rust",  
        "CLI/TUI"  
    ];  
  
    let mut projects: Vec<&str> = vec![  
        // Rust Projects  
        "Hayasen",  
        "ComChan (Serial Monitor)",  
        "Arduino CLI Interactive", // Except This one  
    ];  
  
    let website: &str = "https://vaishnav.world";  
}
```



-
1. `hayasen` is an Embedded Rust library that supports multiple sensors for Embedded Systems.
 2. It provides a type-safe and unified API to interact with various sensors.
 3. Currently supports **MPU9250** and **MPU6050** IMU's.
 4. Support for more sensors on the way.

Links

1. Github (Main repo) : <https://github.com/Vaishnav-Sabari-Girish/Hayasen>
2. Github (Examples repo) : <https://github.com/Vaishnav-Sabari-Girish/Hayasen-Examples>

Contributions are Welcome

```
[package]
name = "hayasen"
version = "0.0.8"
edition = "2024"
description = "A multi sensor library for Embedded Rust supporting multiple sensors"
authors = ["Vaishnav-Sabari-Girish forgamesonly684@gmail.com"]
license = "MIT OR Apache-2.0"
repository = "https://github.com/Vaishnav-Sabari-Girish/Hayasen"
documentation = "http://vaishnav.world/Hayasen/"
keywords = ["embedded", "sensors", "driver", "no-std"]
categories = ["embedded", "no-std", "hardware-support"]
readme = "README.md"

[dependencies]
embedded-hal = "1.0.0"

[features]
default = []
mpu9250 = []
mpu6050 = []
```

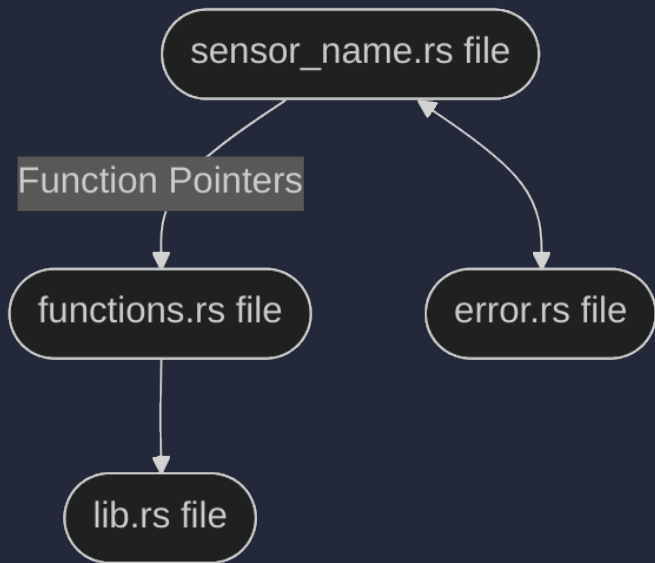
■ Project `src/` directory

```
tree src/
```

[finished]

```
src/
├── error.rs
├── functions.rs
├── lib.rs
├── mpu6050.rs
└── mpu9250.rs

1 directory, 5 files
```



■ Usage - 1

■ Adding the library using `cargo add`

```
cargo add hayasen --features mpu6050 # OR mpu9250
```

■ Adding it to `Cargo.toml`

```
[package]
name = "<project_name>"
version = "0.1.0"
edition = "2024"

[dependencies]
cortex-m = "0.7.7"
cortex-m-rt = "0.7.5"
embedded-hal = "1.0.0"
hayasen = { version = "0.0.8" , features = ["mpu6050"] } # OR mpu9250
nrf52840-hal = { version = "0.18.0", features = ["rt"] }
panic-rtt-target = "0.2.0"
rtt-target = "0.6.1"
```

■ Example program (Generic) for MPU9250

```
use hayasen::mpu9250_hayasen;

fn main() -> Result<(), Error<YourI2cError>> {
    // Setup I2C (platform-specific)
    let i2c = setup_i2c();

    // Initialize sensor with default configuration
    let mut sensor = mpu9250_hayasen::create_default(i2c, 0x68)?;

    // Read all sensor data (temp, accel, gyro)
    let (temperature, acceleration, angular_velocity) =
        mpu9250_hayasen::read_all(&mut sensor)?;

    println!("Temperature: {:.2}°C", temperature);
    println!("Acceleration: [{:.3}, {:.3}, {:.3}] g",
        acceleration[0], acceleration[1], acceleration[2]);
    println!("Angular Velocity: [{:.3}, {:.3}, {:.3}] dps",
        angular_velocity[0], angular_velocity[1], angular_velocity[2]);

    Ok(())
}
```


■ Outputs - 1

■ MPU9250

■

■ Outputs - 2

■ MPU6050

■

■ Why Rust ?

The below points explain why I have used Rust to create `hayasen`

1. Robust **Error Handling** using `Result<>`
2. Memory safety via **Ownership/Borrowing** avoid race conditions and undefined behaviour.
3. Predictable performance from zero-cost abstractions suits real-time constraints.
4. `no_std` support enables bare-metal firmware without OS or `std`.
5. Portable drivers through `embedded-hal` traits across MCU's/HAL's
6. **Ecosystem Gaps** : Some sensor crates are outdated or missing. Which is where `hayasen` comes in.
7. Strong tooling (`cargo`, docs) speeds development and testing.

