Embedded Rust Orientation Detection Application

Author: Reece Wayt (reecwayt@pdx.edu)
GitHub Repo URL: https://github.com/reecewayt/rust-orientation-detection
Description: This project uses the **BBC Microbit V2** which is a SBC with a Nordic nRF52833 processor, on board sensors, leds, buttons, a speaker, and debugger. More specifically, this project will utilize the on-board accelerometer (ST LSM303AGR) to implement smartphone-style orientation detection. This type of orientation detection is fundamental to mobile devices, which enable them to perform step counting, screen rotations, and motion-based gaming. In my application I will be focusing on screen rotations (i.e. state changes) based on the accelerometer data.

Project Features

1. 6D Orientation Detection
    - My application will detect orientation in a 3 dimensional vector space which can be represented as a column or row vector as shown below. Note that these are signed values. $$ \begin{bmatrix} X \ Y \ Z \end{bmatrix}, [X ; Y ; Z] $$
2. Low power functionality after inactivity
    - The accelerometer includes functionality to enter into low power mode after inactivity. Low power is very essential for embedded applications so I plan to implement this in my final application by configuring the sensor to enter into low power.
    - With this functionality, I will need to also configure the sensor to activate after a given acceleration threshold has been reached. Once reached the device will wake up and enter full performance mode.
3. Sensor interfacing will be over I2C Bus
    - The SBC is wired such that the Microbit is connect to this sensor via the I2C serial bus.

Software Requirements

1. Development Environment

- Rust toolchain
- `cargo-embed` or `probe-run` for flashing
- ARM GCC toolchain
- `rust-analyzer` for VS Code support

2. Required Crates (as listed in `Cargo.toml`)

- `microbit-v2`: Hardware abstraction for Micro:bit V2
- `cortex-m` and `cortex-m-rt`: Core ARM support
- `lsm303agr`: Accelerometer driver

Project Milestones

1. Environment Setup
2. Sensor Communication

- Establish I2C communication with LSM303AGR
- Read basic acceleration values

3. Orientation Detection

- Implement orientation detection (Vector Math)
- Create visual feedback by printing human readable values over UART (Virtual Com Port)
- Test all possible orientations

3. Power Management

- Implement low power mode after inactivity configuration
- Configure wake-up threshold
- Test power state transitions

Testing Strategy

Rust does not have a unit testing framework for embedded systems, at least not in the sense of the traditional `#[cfg(test)]` attribute. With that said, I'll be utilizing debuggers and conditional compilation to verify code on the host machine as described below.

1. Debugging

- Utilize VS Code's integrated debugger with custom `launch.json` configurations for JLink connections with Microbit
- RTT (Real-Time Transfer) for debug and error logging at runtime
- Logic Analyzer for reading I2C transactions

2. Conditional Compilation Traits

- Use attributes to enable debug features only in development builds

```
// Only compiles in debug builds
#[cfg(debug)]
print_info(some_info);

// Also include assertions in debug builds
#[cfg(feature = "assertions")]
assert_equal(x, y);
```

Challenges

- Low power mode testing will be difficult and hard to verify especially since the power traces on the board are covered by a silk screen
- I might need to implement interrupt mechanisms
- Depending on the sensor and the stress its been under since manufacturing, I will need to consider error and offset of sensor data.

- Some form of filtering of the sensor data might be needed for clean state transitions