Telemetry

Emily Horsman <horsmane@mcmaster.ca>, Tanner Ryan <ryant3@mcmaster.ca>

April 2020

Preamble

Brief

Telemetry saves significant development time for common embedded applications.

There is a large market of hardware peripherals which use an I^2C bus to communicate with a computer. They usually interface with an embedded device.[1] These embedded controllers need to be flashed to change application logic and this often requires physical access. There are thousands of such peripherals including sensors that measure temperature, moisture, colour, current, etc. Hobbyists and industrial prototypers buy these sensors from distributors such as Adafruit and Sparkfun. These distributors often share libraries that allow these sensors to be used with common embedded controllers with little development time.[2] However, these libraries have a lot of redundancy amongst them and often contain technical tradeoffs.[3]

Writing a program for embedded controllers that reads and writes to I^2C-based hardware peripherals traditionally involves three tasks:

1. Getting data off the I^2C bus with sensor-specific protocols running in an "application layer".
2. Turning raw bytes from the sensor protocol into semantically sensible values. i.e., from bytes to a temperature reading in degrees Celsius.
3. Managing multiple sensors connected to the same I^2C bus which are read and written to at various intervals.

Telemetry is a library for embedded devices that builds on an abstraction of hardware peripherals to simplify these tasks.

1. Generalizes interactions on the I^2C bus to allow declarative configurations for new sensors to be written instead of application logic.
2. Decouples the application logic which deals with data semantics from the embedded device and its development environment. Developers will write a small program independent of the

[1] "Arduino-compatible" microcontrollers are popular choices amongst hobbyists and industrial prototypers.
[2] e.g., For the SHT31 temperature and humidity sensor: `https://github.com/adafruit/Adafruit_SHT31`
[3] e.g., calling functions which sleep the device instead of using an event loop that would allow other tasks to be performed.

firmware flashed to embedded devices without needing to deal with low-level details. These programs can be written in any language and run on servers, instead of being written in the low-level systems languages that embedded devices typically use.[4]

3. Manages peripherals on an I^2C bus without writing additional firmware logic. This includes logic for automatically discovering and processing data when a sensor is physically connected.

This achieves meaningful outcomes for development teams.

1. Much less development time to work with data from sensors. Reduces prototyping to a "plug-and-play" experience.
2. Reduces skill specialization required for embedded development. Application logic can be written in any language without knowledge of embedded C++.
3. Telemetry can be used as a pre-packaged firmware that can be uploaded to a compatible embedded device without changes. This allows you to physically connect sensors and immediately store data and consume data from a provided API.

Emily Horsman and Tanner Ryan are responsible for this aspect of the project.

Software Requirements Specification

Software Design Description

Code Guide

[4]C and C++ are common choices for embedded controllers. Developers can use anything — even something like Haskell — for Telemetry .