Milan SOC-MCU communication

Author: Rohan Mehta
Last Updated: Jan 29, 2020
Status: RFC

Note (2/3/20): this doc needs an update after chatting with Ben Tankersley. We may be able to reuse most of what Stella does, which is quite similar to the diagrams here, but more sophisticated.

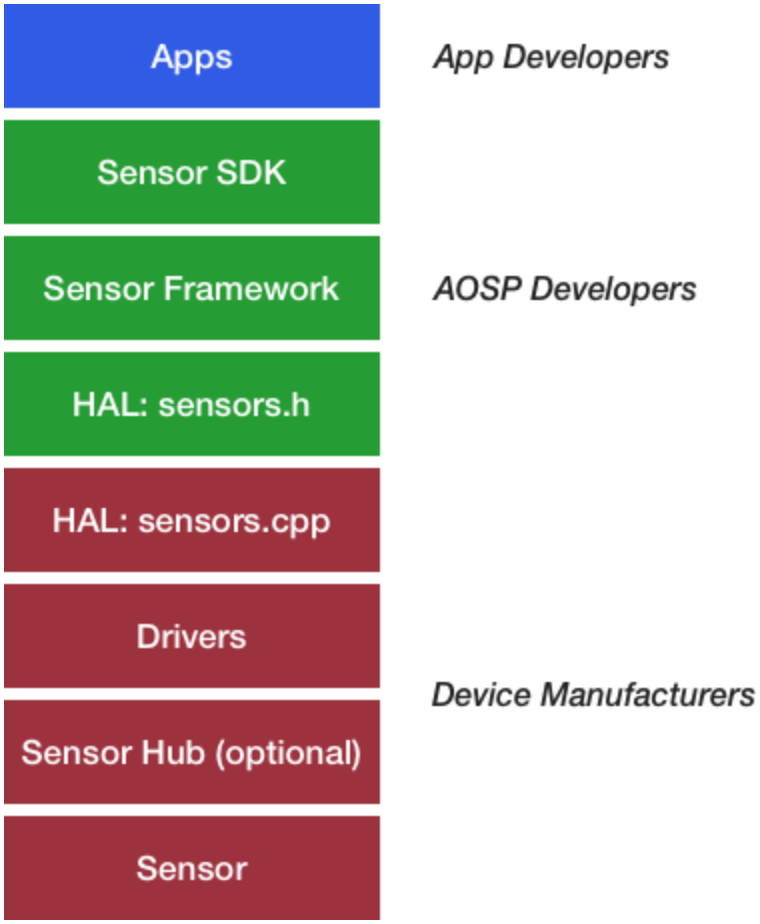
For Milan, we're gonna have:

- MCU running FreeRTOS, that is collecting data from sensors
- SOC running Android, which will want access to sensor data and the ability to change characteristics of the sensors (frequency, latency etc)
- An SPI bus connecting the two

I explored what the architecture for this might look like.

Android Sensor Stack

This is what the Android sensor stack looks like:



Going top to bottom:

- **SDK**: Basically SensorManager.java, which allows app devs to request sensor data
- **Framework**: Handles multiplexing. Links many apps that need sensor data to the HAL (HAL is single-client).
    - Framework handles setting the correct sampling frequency+max latency, based on all the requests from apps. Also handles deactivating sensors when no one is subscribed
    - An implication here is that if an app requires data at a given frequency, it may receive events more often if another app requests things faster. Similar logic for latency.
- **HAL**:
    - Sensor.h is part of AOSP, and lists every type of sensor and some data structures. Also provides APIs for getting list of sensors, activating, setting sampling frequency etc
    - Sensor.cpp is written by vendor, it just implements the interface from the header.
- **Drivers**: Self-explanatory. Interacts with device.
- **Sensor Hub**: A device that can do some low power processing(e.g. a microcontroller)

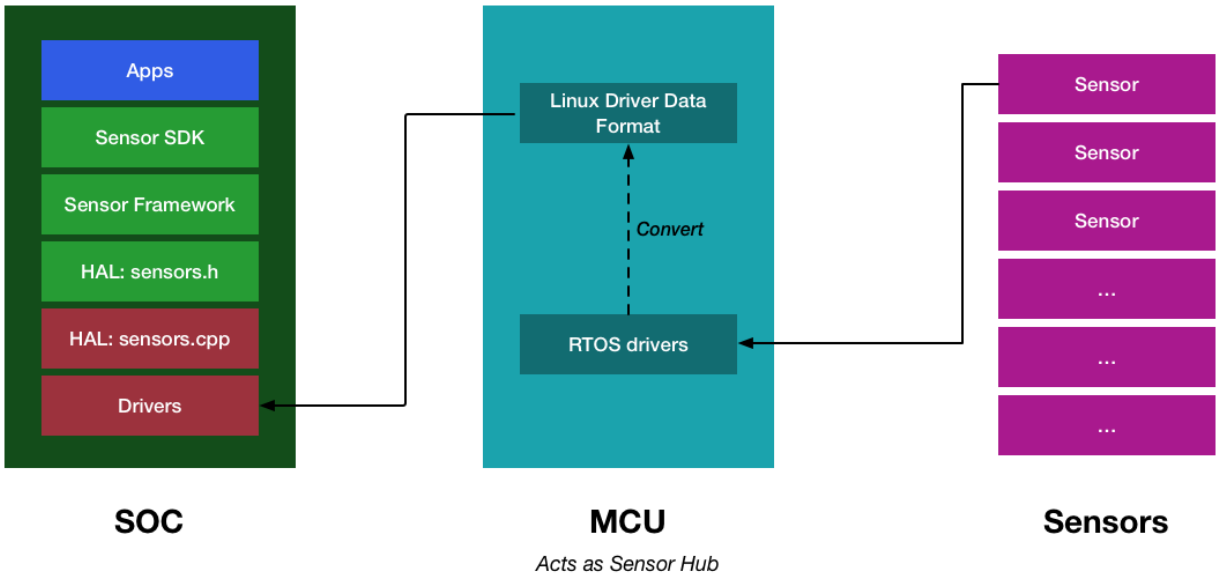
- **Sensor:** the hardware. *Android does not care how the sensor (or sensor hub) is connected. SPI is noted as a common way.*
    - SOC can be in three states: on, idle (medium power), suspend (low power)
    - A sensor can be be one of:
        - **non-wake-up sensor**: do not prevent SOC from going into suspend, do not wake up SOC to report data.
            - Drivers are not allowed to hold wake locks
            - Apps should keep a partial wake lock in case they need sensor data from these sensors when screen is off
            - in suspend mode, sensors should keep working and generating events. These are placed in hardware FIFO
            - FIFO events are delivered when SOC wakes up. Old data is dropped if necessary.
            - latest event is saved outside of FIFO so never lost
        - **wake-up sensor**: these make sure data is delivered irrespective of the SOC state
            - They let SOC go into suspend mode, but wake it up when new data is ready. Specifically, wake it up either (1) before hardware FIFO gets full or (2) max reporting latency expires
            - Also, the driver should hold a 200ms timeout wake lock, to allow apps to respond to the events (*this requirement is supposed to go away in future Android releases)*
    - In general, it is recommended to have one wake-up and one non-wake-up sensor defined for each type. SensorManager code tells you which one is the default one for a given sensor type (generally non-wake-up with a few exceptions)

Proposal

Given the way the Android stack is set up, it seems clear that:

- The MCU can act as the Sensor Hub
- Using SPI, the MCU should talk to the Linux drivers on the SOC

My proposed architecture is this:



The relevant pieces of software are:

- **FreeRTOS drivers:** Written by JDM
- **RTOS->Linux data conversion:** Written by JDM. Ric believes they probably already have this layer, and that it would be cheap.
- **SOC Drivers (Android/Linux drivers):** Written by JDM
- **Sensors.cpp:** Written by JDM.

This means that the Android stack remains exactly as AOSP expects it, which means we get to reuse most of the stack for free. Thoughts?

FAQs

Q: How will we be able to control sampling frequency and other sensor configs from the SOC?
A: This is pretty well defined by android since Sensor Hubs are a core concept. In short, the HAL interface has methods to set these properties. The implementation will route this through the driver, over SPI and to the hub (which is the MCU).

Q: What about BlueBox?
A: BlueBox is just a protocol for sending data. They don't deal with things like drivers. We may still be able to use BlueBox for the SPI transport, though I don't know for sure.

Appendix

Raw notes from researching Android's sensor stack. Feel free to ignore this section.

Sensor Stack

- Sensors -> Sensor Hub -> Drivers -> HAL (sensors.h/cpp) -> Framework -> SDK -> Apps
- SDK = SensorManager.Java
- Framework = multiplexing. Links several apps to the HAL (HAL is single-client). Framework handles setting the correct sampling frequency+max latency, based on all the requests from apps. Also handles deactivating sensors when no one is subscribed
    - Implication here is that if an app requires frequency X, it may receive events more often if another app requests things faster. Similar logic for latency.
    - Framework also contains a default "Sensor fusion" provider, which e.g. can fuse accelerometer/gyro/magnetometer to provide a gravity sensor, etc. The default implementation isn't maintained and runs on the SOC (and hence is power inefficient), so manufacturers are expected to produce their own
- HAL: see below
- Drivers: Interacts with device.
- Sensor Hub: this will be our MCU. Recommendation is to have two interrupt lines going from the sensor hub to the SoC: one for wake-up interrupts (for wake-up sensors), and the other for non-wake-up interrupts (for non-wake-up sensors).
    - Batching: Before delivering to HAL, there may be batching in the hub or in the hardware FIFO

Sensors

- SOC can be in three states: on, idle (medium power), suspend (low power)
- A sensor can be be one of:
    - **non-wake-up sensor:** do not prevent SOC from going into suspend, do not wake up SOC to report data.
        - Drivers are not allowed to hold wake locks
        - Apps should keep a partial wake lock in case they need sensor data from these sensors when screen is off
        - in suspend mode, sensors should keep working and generating events. These are placed in hardware FIFO
        - FIFO events are delivered when SOC wakes up. Old data is dropped if necessary.
        - latest event is saved outside of FIFO so never lost
    - **wake-up sensor:** these make sure data is delivered irrespective of the SOC state
        - They let SOC go into suspend mode, but wake it up when new data is ready. Specifically, wake it up either (1) before hardware FIFO gets full or (2) max reporting latency expires

        - Also, the driver should hold a 200ms timeout wake lock, to allow apps to respond to the events (*this requirement is supposed to go away in future Android releases)*
- In general, it's recommended to have one wake-up and one non-wake-up sensor defined for each type. SensorManager code tells you which one is the default one for a given sensor type (generally non-wake-up with a few exceptions)
- Sensor data is an event with:
    - Handle to sensor
    - timestamp for when the event was measured
    - data (depends on the sensor type)
- Sensors have reporting modes:
    - continuous reporting (e.g. accelerometer)
    - on-change reporting (e.g. step counter)
    - one-shot (report and then deactivate e.g. signification motion detector)

Sensor HAL

- A sensor is a virtual device that provides data. List of sensors in `sensors.h`, which is the Sensor HAL
- Sensor.h lists all the sensors. Interestingly, mic, camera and touch screen aren't listed here — they are considered to be different b/c they are higher bandwidth
- Android doesn't care how sensor is connected: I2C, SPI etc all good. Often there are sensor hubs, which can do some compute without waking up SOC
    - Sounds like our MCU can act as the sensor hub
- HAL impl reports all sensors present on device
    - There can be more than 1 sensor per type
- Device manufacturers (i.e us/JDM) should be filling in sensors.cpp
- Android 10 has updates (sensor Hal 2.0, Multi-Hal 2.0) but we can't use these.
- main functions in HAL are:
    - get_sensors_list
    - activate
    - batch (sets params like sampling freq or max latency)
    - setDelay (sets sampling freq)
    - flush (flushes FIFO of a given sensor)
    - poll (get events)

Sensor

- Has a type (the list of types are in sensors.h)
- If we have new types of sensors, we can add temporary types for them, but we'd eventually want to document

- From the Android perspective, every sensor is independent. So e.g. step counter and accelerometer may share the same physical sensor, but they are considered independent entities by the software
    - Implication is that turning off step counter doesn't necessarily turn off the hardware (e.g. if accelerometer is being used)