Low Level Sensor Programing and Security Enforcement with MRAA

Brendan Le Foll <brendan.le.foll@intel.com>
Agenda

- What’s mraa/mraa.io/libmraa
- Why / What / Where
- Mraa on peripheral manager (PIO)
- Mraa with AGL’s AFB
- Future
Mraa.io - 101

- Simple userspace I/O protocol control for:
  - Gpio, i2c, SPI, ADCs, PWM, UART, 1-Wire
- Made for monkeys – easier is better
- Bindings for C, Python (2 & 3), node.js, & java
- Supports a large number of development boards, platforms, daughterboards & expansion boards
void set_gpio_high(int num) {
    mraa_gpio_context gpio;
    gpio = mraa_gpio_init(num);
    mraa_gpio_dir(gpio, MRAA_GPIO_OUT);
    mraa_gpio_write(gpio, 1);
    mraa_gpio_close(gpio);
}
mraa::Gpio* gpio0 = new mraa::Gpio(8);
gpio0->dir(mraa::DIR_OUT);
gpio0->write(1);
from mraa import *
x = Gpio(8)
x.dir(DIR_OUT)
x.write(1)
x = "memory is not my problem!"
from mraa import *
def hello():
    print("You do not talk about the GIL")

x = Gpio(8)
x.dir(DIR_IN)
x.isr(EDGE_BOTH, hello)
Why?

- Unified API
- Demo of http://mraa.io/demo
Sensor Library – “UPM"

- **http://upm.mraa.io**
  - Sensor browser/integration
  - Documentation / quirks
  - Links to datasheets etc...
- Mostly C++ (some C), java, node.js, python
(some) supported boards
Adding a board

- There are 3 ways to use your board with mraa
  - **Raw mode**, without a platform configuration you can use mraa but no pinmuxing or platform configuration will be done or known. However you can still access your platform IO and this is very useful for testing as well as writing a new platform configuration.
  - **C platform configuration**, there are many examples of how to add a mraa_board_t and load it, it’s surprisingly easy and many companies & individuals have managed to do it on their own unassisted by ourselves. Depending on platform complexity a mraa_board_t must be complemented by an advanced function array which can override mraa’s default IO functions. This is very useful in complex platforms such as Edison or where there are hardware oddities to overcome. Mraa can either be configured with just one platform or with all platforms from that architecture.
  - **Json file**, mraa configuration files for ‘simple’ platforms will be able to provide everything via a json file that is loaded via mraa_load_platform("myconfig.json") using the same configuration format as imraa.
Android Things (a.k.a Brillo/Weave)

Libmraa w/periphmraa backend

mraajava

Application (java)

Binder

Peripheralmanager

Linux Kernel
libmraa + AFB
The *binder* provides the way to connect applications to the services that it needs.
It provides a fast way to securely offer APIs to applications written in any language and running almost anywhere.
The *binder* is developed for AGL.
The *binder* is the usual name.
The binary is named *afb-daemon*.
The name *afb-daemon* stands for *Application Framework Binder Daemon*.
The word *daemon*, here, denote the fact that the *binder* makes witchcraft to connect applications to their expected services.
(note: that usually the term of daemon denotes background process but not here).
Each *binder afb-daemon* is in charge to bind one instance of an application or service to the rest of the system, applications and services.
Within AGL, the connection between services and/or applications is tuned by the AGL framework and the AGL system.
Libmraa + AFB architecture

Libmraa w/AFB backend

Binder
afb-daemon

af-mraa-binding

Application

Security Context

Libmraa (normal)

Linux Kernel
Integrating AFB as a backend

• Mraa has support for platforms to override default behaviour, we do this for odd platforms (Quark X1000), old non standard kernels (Rpi) or because we’re not talking to the linux kernel directly at all (firmata, d2xx, peripheralmanager)

• The AGL ‘platform’ in mraa is compiled in with the –DBUILDARCH=“AFB”, this will trigger all future calls to mraa calls to go through to the AFB bus to the af-mraa-binding.

• The AGL ‘platform’ overrides all functionality in mraa so mraa_i2c_init no longer does anything with /dev/i2c-n

• Synchronizes \texttt{afb_wsj1_call_s} calls

\begin{tikzpicture}
    
    \node[draw,fill=cyan!30] (i2c_mraa_init) at (0,0) {\texttt{i2c_mraa_init()}}; 
    \node[draw,fill=cyan!30] (board_i2c_mraa_init) at (4,0) {\texttt{board\textgreater{}i2c_mraa_init()}}; 
    \node[draw,fill=cyan!30] (board_i2c_init) at (8,0) {\texttt{board_i2c_init()}}; 

    \draw[->,cyan!30] (i2c_mraa_init) -- (board_i2c_mraa_init); 
    \draw[->,cyan!30] (board_i2c_mraa_init) -- (board_i2c_init); 

\end{tikzpicture}
af-mraa-binding

- Af-mraa-binding initializes a ‘normal’ libmraa build against a real `--DBUILDDARCH`. All the build architectures are supported.
- Therefore you need two builds of libmraa, one to link against af-mraa-binding and another to link to your application.
- Uses `wrapjson` from `afb-utilities`.
Future

- Currently the AGL platform in mraa has no knowledge of what is actively supported so querying the platform with mraa-i2c or mraa-gpio does not work, the underlying platform API has to be called for this
- The binding is simplistic, it only exposes two verbs, “dev-init” to initialize a mraa context, and “command” to run a mraa command against an initialized context
- Mraa calls are mostly all blocking so calls to afb_wsj1_call_s require a lock to grab the response, currently some calls assume that the action is successful rather than really checking the response from the hardware
- Only some of mraa’s functionality is supported 100% (i2c)
More info

- http://github.com/intel-iot-devkit/mraa
- upm.mraa.io
- IRC: Freenode #mraa
- ML: mraa@lists.01.org