Using Zephyr, Linux and Greybus for IoT

Chris Friedt

chris@friedt.co
The Internet of Things

“A system of interrelated computing devices, mechanical and digital machines provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.”

-- Wikipedia
Popular IoT Devices
Popular IoT Devices

- /dev/gpiochipN
- /dev/i2cN
- /dev/videodevN
- /sys/bus/iio/devices/iio:deviceN
...
Discoverable Buses

1. Must be able to query devices attached to the bus
2. Services provided by each device must be categorized
3. Services should use a standard protocol

PCle, USB, Ethernet are discoverable
GPIO, I2C, SPI are non-discoverable
Project Ara

- Motorola Advanced Technologies and Projects (ATAP)
- Former DARPA
- Acquired by Google
Project Ara

- Modules could use GPIO, I2C, SPI, MIPI, etc.
- UniPro High-speed Interconnect
- Shelved in 2016
Due to the application-layer protocol being used on top of UniPro for Project Ara, all of those non-discoverable buses that we had been using in embedded for so long had suddenly become discoverable*.

*with some exceptions
Greybus
[manifest-header]
version-major = 0
version-minor = 1

[interface-descriptor]
vendor-string-id = 0x1
vendor-product-id = 0x2

; Interface vendor string
[string-descriptor 0x1]
string = Zephyr Project RTOS

; Interface product string
[string-descriptor 0x2]
string = Greybus Service Sample Application

; 'Control' class on Bundle 0
[bundle-descriptor 0x0]
class = 0x0

; 'Bridged PHY' class on Bundle 1
[bundle-descriptor 0x1]
class = 0xa

; 'Bridged PHY' class on Bundle 2
[bundle-descriptor 0x2]
class = 0xa

; 'Bridged PHY' class on Bundle 3
[bundle-descriptor 0x3]
class = 0xa

; 'Control' protocol on CPort 0
[cport-descriptor 0x0]
bundle = 0x0
protocol = 0x0

; 'I2C' protocol on CPort 2
[cport-descriptor 0x2]
bundle = 0x2
protocol = 0x3

; 'SPI' protocol on CPort 3
[cport-descriptor 0x3]
bundle = 0x3
protocol = 0xb
Greybus Special Entities

AP (Applications Processor)
- Lives inside the Linux kernel
- Communicates via UniPro using AP Bridge
- Administrates the Greybus network

SVC (Supervisory Controller)
- Notify AP when modules are inserted or removed
- Configure and Control the transport (e.g. UniPro)
Gbridge: Greybus for IoT

- Alexandre Baillon, BayLibre
- **ELCE 2016, Berlin**
- **Plumbers 2019, Lisbon**
- Implement SVC in software
- Communicate with AP (Linux kernel) via Netlink
- gb-netlink kernel module

[GitHub Repository](https://github.com/anobli/gbridge)

[_netlink Module](https://github.com/anobli/greybus/tree/gb_netlink)
Project Ara Topology
Gbridge Topology
Gbridge Host Controllers

- UART (CPort buried in reserved bits)
- BLE (using Generic Attribute Profile?)
- GB Sim (simulated Greybus module using BBB)
- TCP/IP (mDNS discovery, CPPorts as seq TCP ports)
Gbridge Host Controllers

- UART (CPort buried in reserved bits)
- BLE (using Generic Attribute Profile?)
- GB Sim (simulated Greybus module using BBB)
- TCP/IP (mDNS discovery, CPorts as seq TCP ports)

Focus on TCP/IP for the simple reason, that all Greybus requires is a reliable transport
Features:

- BSD Sockets API
- Network protocols
- 6LowPAN
- IP over everything
- POSIX threads
- Device Tree

- Kconfig
- Menuconfig
- Shell!
- Documentation!
- Community!
Zephyr Menuconfig

- $ ninja -C build menuconfig
- familiar look & feel
- easily find Kconfig symbols and info
Zephyr & Device Tree

In board directory. The “base” devicetree, includes .dts files.

Set by DTC_OVERLAY_FILE
Optional DTS format files which override BOARD.dts.

FILE_1.overlay
FILE_n.overlay

In zephyr/dts/bindings/:
Extensible with DTS_ROOT__
Contain rules for macro generation from devicetree.

Intermediate output in build/zephyr/. Combination of <BOARD>.dts and overlays.

dtc compiler, just to catch errors/warnings

Final merged devicetree in build/zephyr/. Useful for debugging and understanding.

Final output in build/zephyr/include/generated-
Generated C headers abstracted and accessed via <devicetree>
The original NuttX Greybus implementation had relatively few runtime and OS requirements:

- POSIX threads
- malloc / heap allocation
- unistd.h sleep routines
- qsort
- atomics
- linked-lists
Porting Greybus to Zephyr

- Port done last year
- Sneak preview of Leash PCB
- Blinking LEDs
- UART only
- Demo hardcoded a lot of things
Zephyr & Device Tree

● DT is the single source of all hardware configuration
  ○ Includes virtual hardware
● DT bindings compatible with Linux & other OS’s
● Preprocessed DT used to generate C header
  ○ All DT information is compile-time const
  ○ Unused DT nodes occupy 0 bytes in ROM
  ○ Cannot parse DT at runtime
Zephyr Devices

- Devices instantiated by macro
  ```c
  #define DT_DRV_COMPAT my_dt_label
  #include <devicetree.h>
  #define MY_FUN(_num) ...
  DT_INST_FOREACH_STATUS_OKAY(MY_FUN)
  ```
- All Zephyr devices have
  - Mandatory init function (ro)
  - optional config (ro), data (rw), api (ro)
Integration Options:

1. 🗯 require users to maintain a Greybus Manifest AND Device Tree AND ensure that the two are consistent

2. 🇺🇸 somehow generate Manifest from Device Tree or vice-versa
Greybus Manifest ≠ Device Tree 👎

- Insufficient platform metadata to go in this direction
  - Implies that 10’s (100’s?) of board-specific init functions exist
- Possibly requires runtime parsing of DT
  - Not possible in Zephyr
- Possibly requires handling permutations of enabled device instances
  - Undesired ROM footprint
Device Tree => Greybus Manifest 👍

- DT is a Superset of what is required for Manifest
- Conveniently handles nested relationships
  - Possibility of supporting multiple, isolated Greybus instances
- Handle the base case for a given platform
  - Allow User Manifests (e.g. in EEPROM) to override default given a fixed pinmux / pinconf
Greybus (DT)

```c
#include <dt-bindings/greybus/greybus.h>
/

greybus0: greybus0 {
    compatible = "zephyr,greybus";
    label = "GREYBUS_0";
};
};
&greybus0 {
    status = "okay";
    version-major = <GREYBUS_VERSION_MAJOR>;
    version-minor = <GREYBUS_VERSION_MINOR>;
    /* … interfaces … */
};
```
Greybus String (DT)

&greybus0 {
    /* ... */
    gbstring1: gbstring1 {
        status = "okay";
        compatible = "zephyr,greybus-string";
        /* string id 0 is invalid */
        id = <1>;
        greybus-string = "Zephyr Project RTOS";
    };
    /* ... */
};
Greybus Interface (DT)

&greybus0 {
    /* ... */

    gbinterface0 {
        status = "okay";
        compatible = "zephyr,greybus-interface";
        /* give phandle rather than integer */
        vendor-string-id = <&gbstring1>;
        product-string-id = <&gbstring2>;
        greybus-interface;
    }
    /* ... */

};
/* ... */
};
Greybus Bundle (DT)

&greybus0 {
    /* ... */
    gbbundle0 {
        status = "okay";
        compatible = "zephyr,greybus-bundle";
        id = <42>; /* 0 is reserved for the Control Bundle */
        bundle-class = </* fixed based on child CPorts */>;
        /* ... CPorts nested inside of bundles ... */
    }
    /* ... */
};
&greybus0 {
    gbbundle0 {
        /* ... */
        bundle-class = "<BUNDLE_CLASS_BRIDGED_PHY>";
        gbgpio0 {
            status = "okay";
            compatible = "zephyr,greybus-gpio-controller";
            greybus-gpio-controller = "<gpio0>";
            id = <1>;
            cport-protocol = "<CPORT_PROTOCOL_GPIO>";
        }
    }
};
I2C CPort (DT)

```c
&greybus0 {
  gbbundle0 {
    /* ... */
    bundle-class = <BUNDLE_CLASS_BRIDGED_PHY>;
    gbi2c0 {
      status = "okay";
      compatible = "zephyr,greybus-i2c-controller";
      greybus-i2c-controller = <&i2c0>;
      id = <1>;
      cport-protocol = <CPORT_PROTOCOL_I2C>;
    }
  }
};
```
gbspi0 {
    status = "okay";
    compatible = "zephyr,greybus-spi-controller";
    greybus-spi-controller = <&spi0>;
    id = <1>; /* CPort ID */
    cport-protocol = <CPORT_PROTOCOL_SPI>;
    /* Entries for struct gb_spi_master_config_response */
    bpw-mask = <0xff>;
    min-speed-hz = <2000000>;
    max-speed-hz = <6000000>;
    mode = <0>;
    flags = <0>;
    /* greybus spi devices */
SPI Device (DT)

gbspi0 {
    gbspidev0 {
        status = "okay";
        compatible = "zephyr,greybus-spi-peripheral";
        cs = <0>; /* used as gpio array index in spi phandle of parent device */
        /* Entries for struct gb_spi_device_config_response */
        mode = <0>;
        bpw = <8>;
        max-speed-hz = <8000000>;
        device-type = <GB_SPI_SPI_DEV>;
        device-name = "ADXL362";
    }
};
Fake GPIO, I2C, & SPI

Another core philosophy of Zephyr is to test everything.

Development of DT bindings was mostly driven via test cases using “fake” device interfaces.

```
tests/subsys/greybus/gpio/boards
|-- native_posix_64.conf
|-- native_posix_64.overlay
|-- native_posix.conf
|-- native_posix.overlay

tests/subsys/greybus/i2c/boards
|-- cc1352r1_launchxl.overlay
|-- native_posix_64.conf
|-- native_posix_64.overlay
|-- native_posix.conf
|-- native_posix.overlay

tests/subsys/greybus/spi/boards
|-- cc1352r1_launchxl.overlay
|-- native_posix_64.conf
|-- native_posix_64.overlay
|-- native_posix.conf
|-- native_posix.overlay
```
Greybus System Service

- Leveraged Device Tree to automatically generate the manifest.
- Leveraged Device Tree to automatically create virtual
- Why not also automatically start a Greybus Service?
- Not a single line of code required
  - Declare GB resources in Device Tree
  - Support for drivers in Kconfig
Linux Driver Ecosystem

- Keep the intelligence in the host
- Vaishnav Ma has done a great job of re-using existing Linux drivers
- Same driver works regardless of Wireless SoC
- Drivers get updated & maintained in the Linux kernel
Greybus in Linux: a Proposal

● Currently with the Gbridge Topology, the SVC lives in userspace and effectively routes messages between AP and other Greybus devices
● By using a connected socket (or really any file descriptor), we can move the SVC into the kernel
● Gbridge would then stay in userspace as mainly an auth + session broker
Linux: WPANUSB

- being developed further as more of a generic interface to IEEE 802.15.4 USB hardware
  - Enable Linux & RTOS devs, as well as HW manufacturers
  - Originally developed by Andrei Emeltchenko (Intel)
  - Already supported in Zephyr (less extensions below)
  - Commitment to add support in RIOT OS (Koen Zandberg)
- Add GET_EXTENDED_ADDR command
- Add GET_CAPACITYILITIES command
- Set LBT in USB
- Set frame retries in USB (for controllers lacking auto-ack)
Open Problems: Greybus

- authentication: support multiple mechanisms
  - public key auth (in progress)
- encryption: standardize on one or two methods
  - aes-128 (in progress)
- auth negotiation
- commissioning, joining, rejoining
- cloud: management of devices at scale
Open Problems: Zephyr

- pthread: dynamic stack allocation (in progress)
- dt-bindings: greybus: mask for fixed pin direction
- dt-bindings: greybus: pwm, adc, camera, ..
- ieee802154: cc1352r: TI RF driverlib (in progress)
- ieee802154: cc1352r: subghz (in progress)
- ble: cc1352r: split-stack driver (in progress)
- proper Zephyr module repo for Greybus
- init: dt: deterministic module loading (in progress)
Open Problems: Linux

- gbridge: discovery should be ip-version agnostic
- gbridge: move svc in-kernel
- greybus: pass fd + session key to kernel post auth
- wpanusb: feature implementation, test, & lkml patch
- opt3001: specify i2c bus / addr with modprobe
## Additional Resources

Current development branch for Greybus in Zephyr

https://github.com/cfriedt/zephyr (branch greybus-service-lpc2020)

Zephyr Getting Started Guide / Slack

https://docs.zephyrproject.org/latest/getting_started/index.html

BeagleConnect (Hardware Rev C) / Slack

https://github.com/jadonk/beagleconnec

<table>
<thead>
<tr>
<th>Board</th>
<th>Topic(s)</th>
<th>Instructions</th>
<th>Video</th>
</tr>
</thead>
<tbody>
<tr>
<td>CC1352R1 LaunchXL</td>
<td>GPIO LED IEEE 802.15.4</td>
<td><a href="https://bit.ly/31FQNV4">https://bit.ly/31FQNV4</a></td>
<td><a href="https://youtu.be/hd60CbiUN1g">https://youtu.be/hd60CbiUN1g</a></td>
</tr>
</tbody>
</table>
Conclusion

● Now is the time to start contributing!
● Developers Wanted!! - Zephyr, Linux, Cloud, BeagleConnect
● Greybus itself may well begin to evolve very soon..
Thank You!