EMBEDDED
OPEN SOURCE
SUMMIT

Zephyr® Project
Developer Summit

Greybus for Robotics with Zephyr RTOS

2023-06-28, Prague

Vaishnav Achath

Texas Instruments

About us | TI Processors and Open source

THE LINUX FOUNDATION

Real-Time LINUX

yocto PROJECT

Decades of contribution and collaboration

Ingrained culture to give back to the community

Upstream FIRST!

Focus on long term, sustainable and quality products

Upstream and opensource ecosystem in device architecture

Upstream FIRST mentality!

Speaker | Intro

Vaishnav Achath, Software Engineer at Texas Instruments India. Vaishnav primarily works on Linux Kernel and U-Boot as part of the Texas Instruments Linux development team. Vaishnav is also a maintainer for TI platforms in Zephyr RTOS.

Introduced to Greybus as part of Google Summer of Code project with Beagleboard.org for adding MikroElektronika Click Board Support through greybus in 2019.

Overview

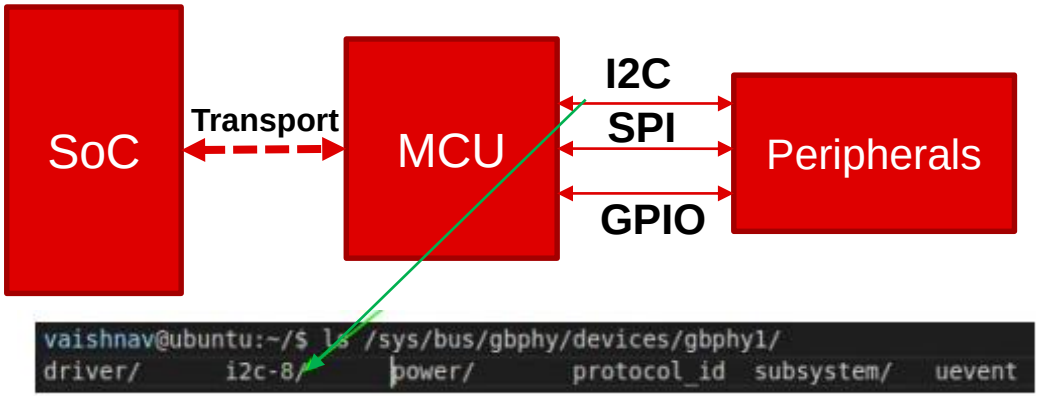
- Greybus.
- Greybus for IoT
- BeagleConnect
- Limitations of Greybus in IoT.
- Greybus Python host.
- Extending Greybus operations.
- Demo.

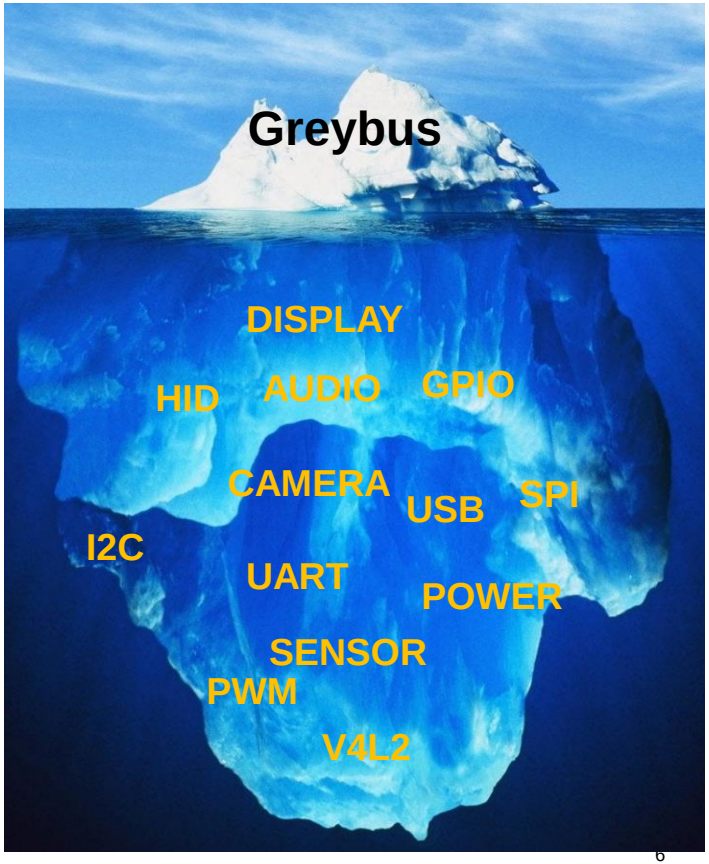
Greybus

- Greybus is an application layer protocol initially designed for Google's Project Ara smartphone and is the same technology being used in Motorola's Moto Mods for modules.
- The central concept behind greybus **is keeping the intelligence in the host**. The external modules interface with the host. The host discovers what is connected, sends instructions (greybus operation), and performs operations on the module, modules respond back in the same protocol.
- Greybus can be thought of as an RPC-like application layer protocol that allows control and interfacing of modules from a host.
- **Greybus is unique because it can interact with various parts of the Linux kernel.**

Credits: Project Ara, Google LLC; Moto Mods, Motorola Inc.

Greybus

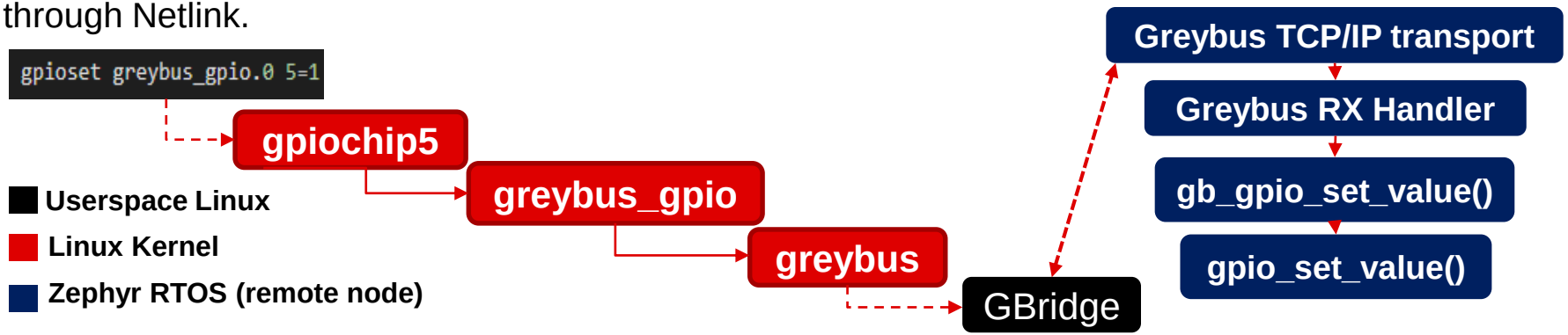




- Greybus allows peripherals on a remote device to appear as if they were on the host through any transport.
- Greybus supports module discovery, and hot-plug/unplug during run-time.
- Allows to keep the intelligence in the host.
- Remote device requires very generic firmware.

Greybus for IoT

- There have been developments in using greybus for IoT applications and also a dedicated Zephyr Module for Greybus exists which allows usage of greybus in remote nodes running Zephyr RTOS.
- Primary transport being used is TCP/IP, even though other transports like UART are implemented.
- On the host side, GBridge is an application that can be used to support greybus over TCP/IP, and the message transfer to the kernel occurs through Netlink.



Credits: Alexandre Bailone, Christopher Friedt, Jason Kridner.

BeagleConnect

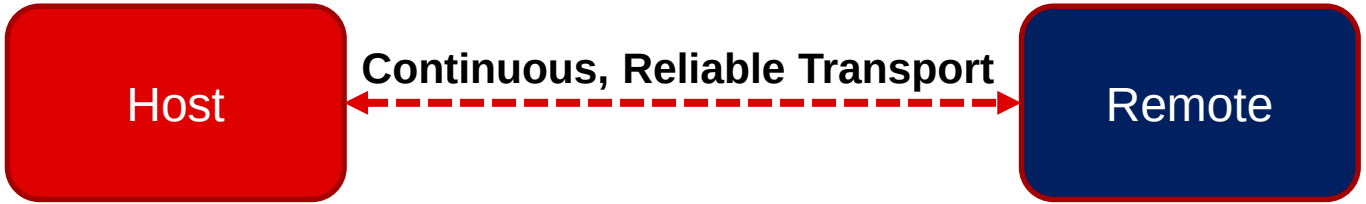
- BeagleConnect™ is a revolutionary technology that virtually eliminates low-level software development for IoT and IIoT applications, such as building, factory, and home automation.
- BeagleConnect™ simply eliminates the need for low-level software development by shifting the burden into the most massive and collaborative software project of all time, the Linux kernel.
- BeagleConnect™ achieves this through greybus as the primary component.

Credits: BeagleBoard.org Foundation

Limitations of greybus for IoT and Real-time Control Applications

Greybus, originally implemented for Project Ara focuses on keeping the intelligence in the host, In a **few** IoT applications, this can limit the use cases because of the following reasons:

- **Host is a single point of failure** – if the host fails or the connection between the host and remote fails, the entire system's functionality is affected.
- For each minute operation there needs to be a transaction over the transport, this **may not be optimal in low-power use cases**.
- Remote node use cases require **real-time control**.



Greybus operation

- The Greybus "connection" is a bidirectional communication path between two “Cports”.
- Each greybus operation is a message transfer from the host to the remote, Each message sent begins with a short header, followed by operation-specific payload data.
- Host generates the greybus operation and dispatches it to the remote, remote node has generic firmware that can perform the corresponding operation(and send back results).

```
struct gb_operation_hdr {
    __le16 size;
    __le16 id;
    __u8 type;
    __u8 result;
    __u8 pad[2];
};

struct gb_gpio_set_value_request {
    __u8    which;
    __u8    value;
} __packed;
```

Greybus Python Host

Currently, greybus testing involves multiple components like **Linux kernel greybus drivers**, **gb-netlink** (Kernel-Userspace), **gbridge**(Kernel-Userspace-Transport), and the **remote node running Zephyr**.

```python
class GBOperation(object):
    OPERATION_HEADER_FMT        = '<HHBBBB'
    MTU = 2048

    def __init__(self, _type, _cport, _id = 0, local = 0):
        self.type = _type
        self.id = _id
        self.result = 0
        self.pad = [_cport, local]
        self.size = 8
        self.operation = bytearray(GBOperation.MTU)

    def payload(self, payload):
        self.operation[self.size:] = bytearray(payload)
        self.size += len(payload)

    def generate(self):
        if self.size % 4 != 0:
            self.operation[self.size:] = [0]*(4 - (self.size % 4))
            self.size += (4 - (self.size % 4))
        struct.pack_into(GBOperation.OPERATION_HEADER_FMT, self.operation, 0,
                 self.size, self.id, self.type, self.result,
                 self.pad[0], self.pad[1])
        return self.operation[:self.size]
```

```python
class GBGPIOOp(object):
    GB_GPIO_TYPE_PROTOCOL_VERSION = 0x1
    GB_GPIO_TYPE_DIRECTION_IN = 0x06
    GB_GPIO_TYPE_DIRECTION_OUT = 0x07
    GB_GPIO_TYPE_GET_VALUE = 0x08
    GB_GPIO_TYPE_SET_VALUE = 0x09

    def __init__(self, cport = 1):
        self.cport = cport
```

```python
class GBGPIO(object):
    def __init__(self, transport, cport):
        self.cport = cport
        self.transport = transport
    def direction_out(self, which, value):
        self.transport.write(GBGPIOOp().direction_out(which, value))
        return self.transport.read(8)[5]
    def direction_in(self, which):
        self.transport.write(GBGPIOOp().direction_in(which))
        return self.transport.read(8)[5]
    def setvalue(self, which, value):
        self.transport.write(GBGPIOOp().setvalue(which, value))
        return self.transport.read(8)[5]
    def getvalue(self, which):
        self.transport.write(GBGPIOOp().getvalue(which))
        return self.transport.read(9)[8]
```

Overcoming limitations | Proof of Concept

- Send a group of greybus operation messages together and execute at remote together.
- Challenge: decision-making and repeating operations at a remote node?
- **Proof of concept**:
  - **Greybus scratchpad region at remote** – host can write/read from this region, and also store response payload of another operation/use this region ad payload for an operation.
    - Two new operations – scratch read and write (write can write constants from the host, or write another operation payload to the scratch region)
    - 1 math operation to perform arithmetic operations on scratchpad region.
  - **Greybus conditional and iteration operations** (if, while) –
    - Conditionally execute a set of operations or loop over a set of operations when the condition is satisfied, all the operands are from the scratch region.
  - **Local Playback Operation** – Playback a group of operations continuously.

Greybus Local Scratchpad

- This is a region of memory in the remote node that the host can read/write through greybus operations.
- Also, another greybus operation response payload can be stored or the data from this region can be used as a payload for another operation.
- For most operations payload is < 4 bytes, so the scratchpad region is modeled as a uint32_t for simplicity.
- Also implements arithmetic/logical operations on the scratchpad region.

```
struct gb_local_memop_request
{
__u8 offset;
__u8 opcount;
__le32 value;
} __packed;

struct gb_local_mathop_request
{
__u8 dest;
__u8 src1;
__u8 src2;
__u8 operator;
} __packed;
```

Greybus Local Decision Making/Loop Operations

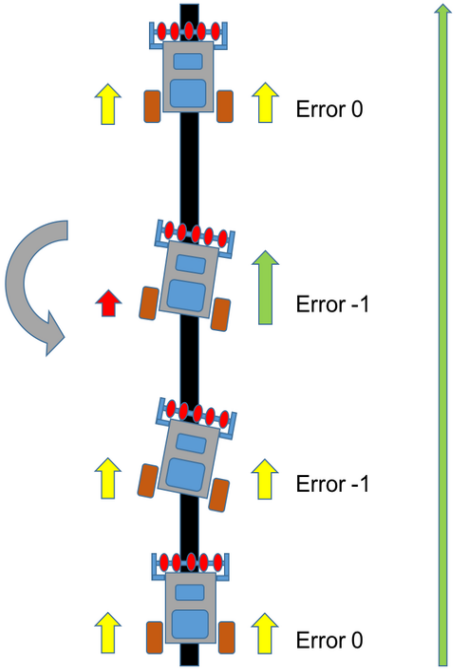
- Provide a minimal set (if, while) of decision-making and loop operations, this helps to perform tasks independently on the remote node along with the scratchpad operations.
- All the condition operands are set from the scratchpad region through a separate operation.

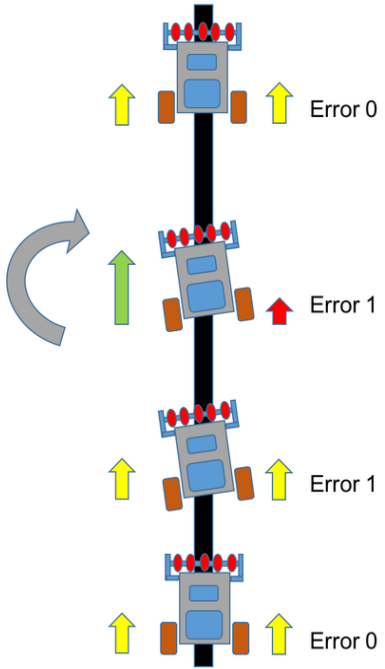
```
struct gb_local_ifop_request
{
__u8 offset;
__u8 opcount;
} __packed;

struct gb_local_whileop_request
{
__u8 offset;
__u8 opcount;
} __packed;
```

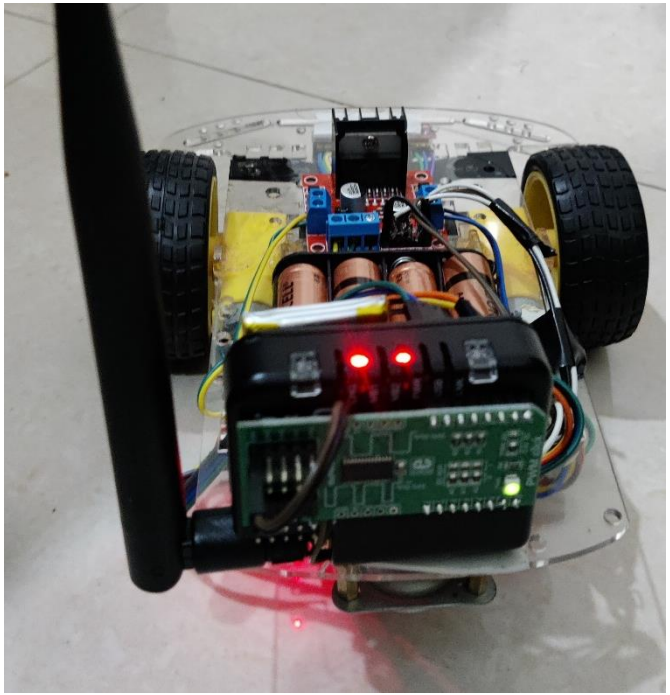
- All new operations use new unique IDs, and also use an unused field in the greybus operation header for indicating it is a local operation.
- Also these operations can accept another operation as a payload.

```
struct gb_local_playback_request
{
    __u8 op_count;
};      You, 3 days ago • core …
```

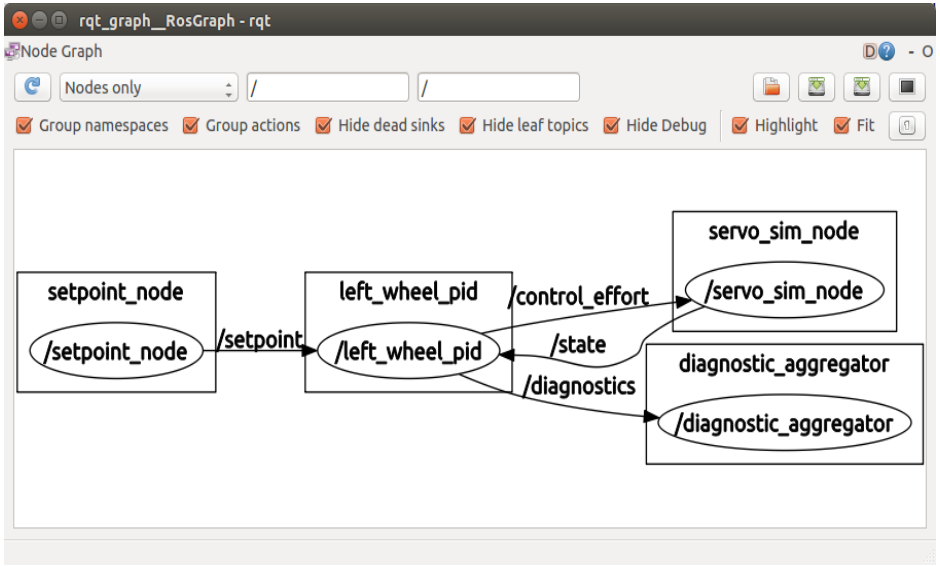
Demo: Line Tracer

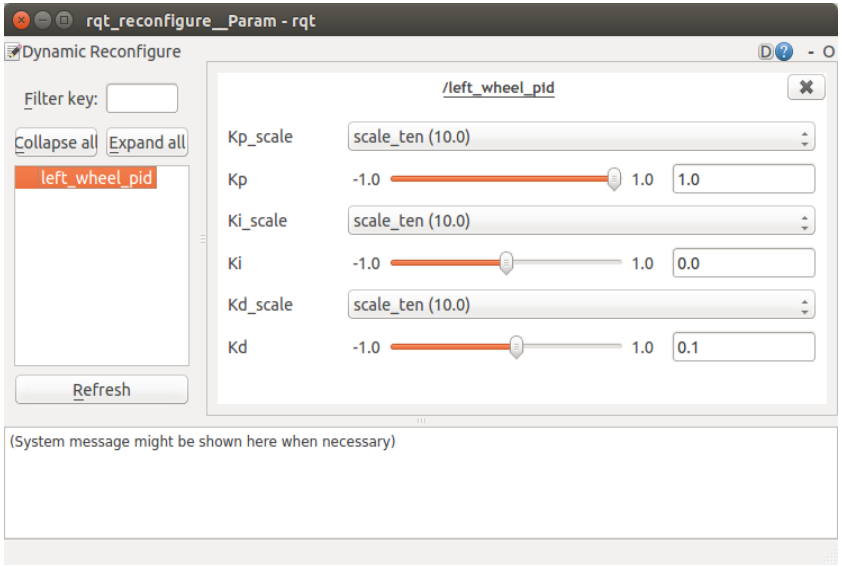






PID Tuning/Control with ROS





Credits: http://wiki.ros.org/pid

Limitations

- Interface available for the user to program is not straightforward.
- Difficult to interface with Kernel – But can be made to work together with Kernel greybus (Kernel performs complex setup and calibration operations, greybus userspace/playback client performs repeated triggered measurements).

References

- MikroElektronika Click Identifier Specs
- BeagleConnect, BeagleBoard.org
- Greybus for IoT, Alexandre Bailone.
- Using Linux, Zephyr and Greybus for IoT, Christopher Friedt, IoT Micro conference LPC 2020.
- Instructables – PID Line follower.
- http://wiki.ros.org/pid

- Details and Instructions:
  - https://github.com/vaishnavachath/eoss23-zdc-greybus

Credits and Acknowledgement

- Jason Kridner, Christine Long, Robert C Nelson, Cathy Wicks, Drew Fustini, Deepak Khatri and BealeBoard.org Foundation community.
- Ivan Rajkovic, Nenad Marinkovic – MikroElektronika.
- Christopher Friedt.
- Nishanth Menon.
- Alexandre Bailone.
- Chris Gammel, Erik Larson.
- Texas Instruments Inc.
- The Linux Foundation.
- Project Ara.

Q&A

- Contact Information:
  - Vaishnav Achath <vaishnav.a@ti.com>
- Also on IRC @ libera.chat #linux-ti

Learn more about TI products

- https://www.ti.com/linux
- http://opensource.ti.com/
- https://www.ti.com/processors
- https://www.ti.com/edgeai

Why choose TI MCUs and processors?

Scalability

Our products offer scalable performance that can adapt and grow as the needs of your customers evolve.

Efficiency

We design products that extend battery life, maximize performance for every watt expended, and unlock the highest levels of system efficiency.

Affordability

We strive to make innovation accessible to all by creating cost-effective products that feature state-of-the-art technology and package designs.

Availability

Our investment in internal manufacturing capacity provides greater assurance of supply, supporting your growth for decades to come.