Qualcomm Developer Network Presents

Developing for Industrial IoT with Embedded Linux OS on DragonBoard™ 410c by Timesys University

Co-sponsored by Qualcomm Technologies, Inc. and
Arrow Electronics

Embedded Linux from a Trusted Source

timesys®

Session 2
Application Development for Embedded Linux®

Maciej Halasz, Vice President of Technology
Timesys Corporation

Webinar Series

- **Session 1:** Introduction to DragonBoard 410 SoC and Starting Development of Your Embedded Linux based “Industrial Internet of Things” (IIoT) Device
  - Setup for designing IIoT products
  - How to assemble and deploy initial BSP
- **Session 2**: Application Development for Embedded Linux
  - Application development environment setup
  - How to reflect product requirements in the BSP
  - Communication in the IIoT system
- **Session 3**: Building a Cutting-Edge User Interface with Qt®
  - Developing modern, rich UIs for factory terminals
- **Session 4**: Embedded Products Security
  - Designing security-rich devices

Recap of the previous session

- **What we did:**
  - Went through the process of flashing rescue and Linux images to the DragonBoard 410c
  - Discussed the high level requirements of the IIoT system
  - Learned about OpenEmbedded Reference Platform Build (RPB) BSP from Linaro™/96Boards™
  - Setup the OpenEmbedded RPB on a host PC
  - Rebuilt the rpb-console-image locally
- **Key takeaways:**
  - OpenEmbedded RPB BSP ≠ Yocto Project® BSP
    - Use same BitBake build engine but different BSP structure
  - Must use a micro SD card to rescue the board
  - Reference images provided by 96Boards
    - Can also receive them from Timesys®
  - Linux support available
    - Forum: 96Boards
    - Ticket-based: Timesys

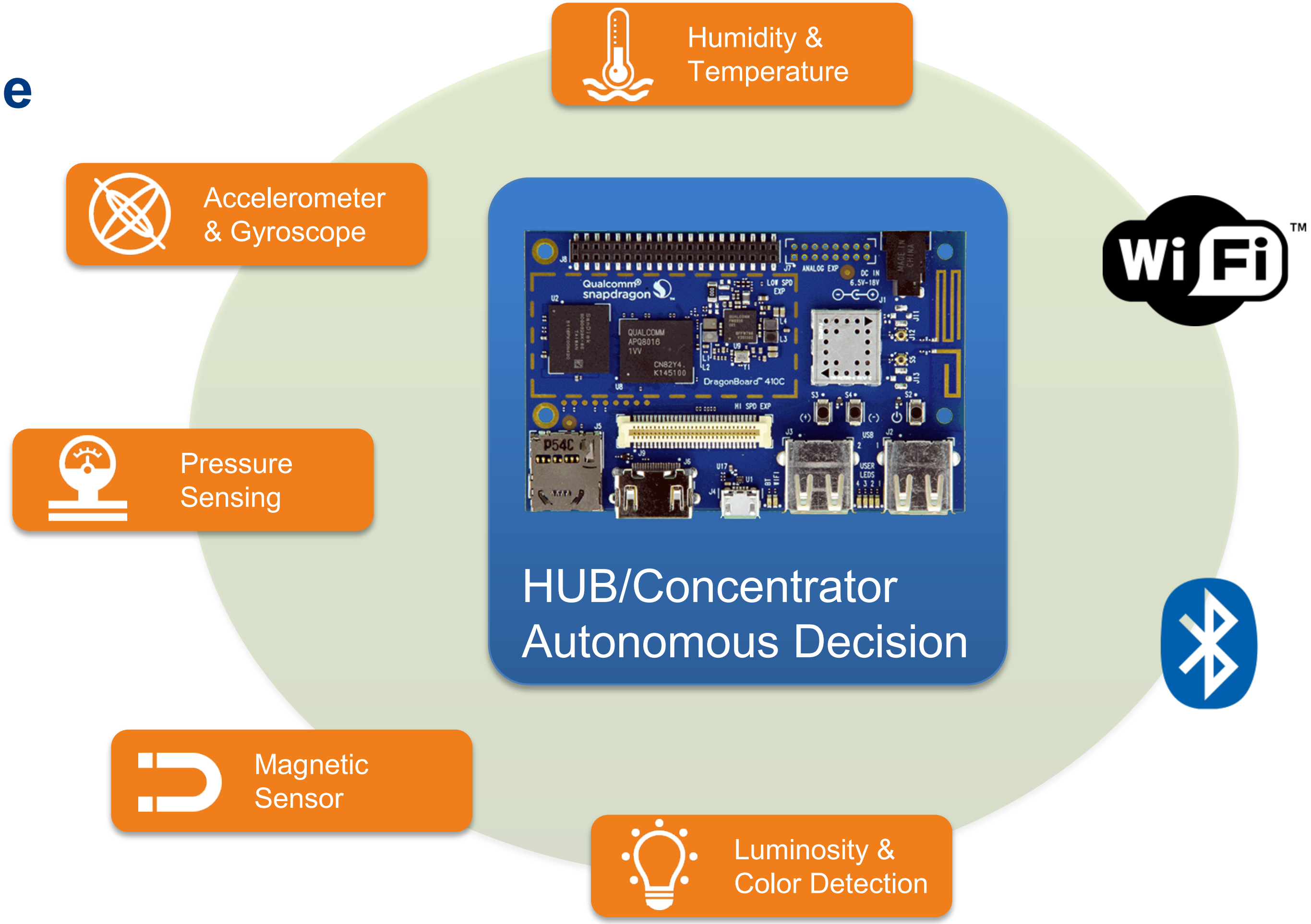
Session 2 — Agenda

- **IIoT Application requirements considerations**
  - Hardware
  - Software
- **Reflecting API requirements in OpenEmbedded RPB Linux BSP**
  - Customizations
- **Application Development Environment setup**
- **Application Development**
  - IDE based C/C++ development/deployment/debugging
  - Data gathering/classification from IIoT end points
  - Application logic development
  - System analysis and tuning
    - Profiling
    - Tracing

Industrial IoT Product Requirements

IIoT point requirements

- Requirements are industry process-specific
- Topography of IIoT can vary
- Requirements typically include
  - Connectivity:
    - Bluetooth®
    - Ethernet ®
    - WiFi™
    - BUS
  - Sensors:
    - Discrete
    - Continuous
  - Sensor examples
    - Temperature
    - Pressure
    - Movement
    - ON/OFF



Humidity & Temperature

Accelerometer & Gyroscope

Pressure Sensing

HUB/Concentrator
Autonomous Decision

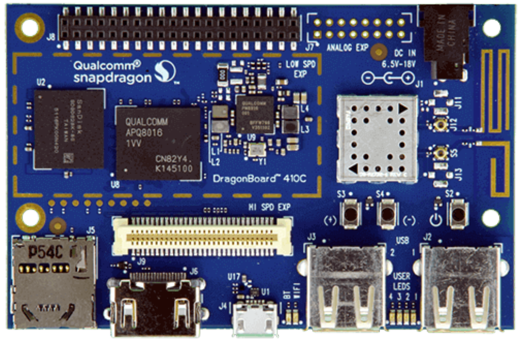
Magnetic Sensor

Luminosity & Color Detection

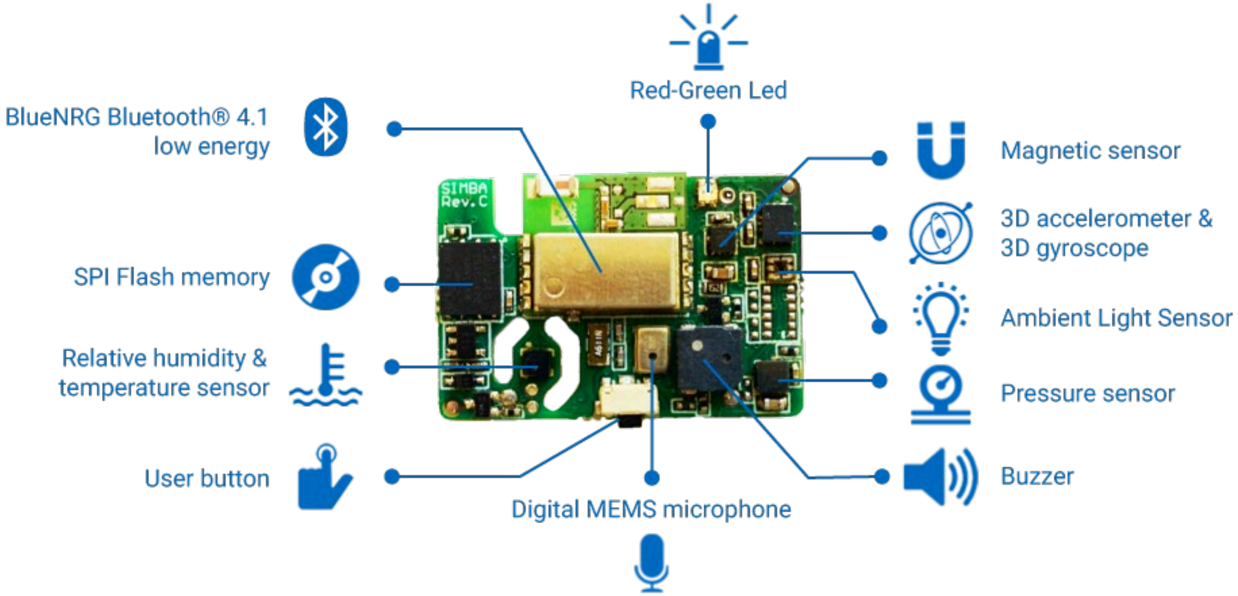
Requirements summary

Hardware

- DragonBoard 410c



- SensiBLE module



Software (DragonBoard 410c only)

- LTS Linux
- Root filesystem APIs:
  - Baseline (Boot Linux)
  - BlueZ (Bluetooth LE)
  - Mosquitto™ (MQTT)
  - OpenSSL ™ + OpenSSH (Secure Shell)
  - Wireless tools (WiFi)
- Application shall be written in C++
- Application shall:
  - Gather data via BLE from SensiBLE sensors
  - Analyze the information and perform an action:
    - Alert – light is too bright
    - Robotic arm is moving in direction XYZ

Reflecting Software Requirements in OpenEmbedded RPB BSP for the DragonBoard 410c

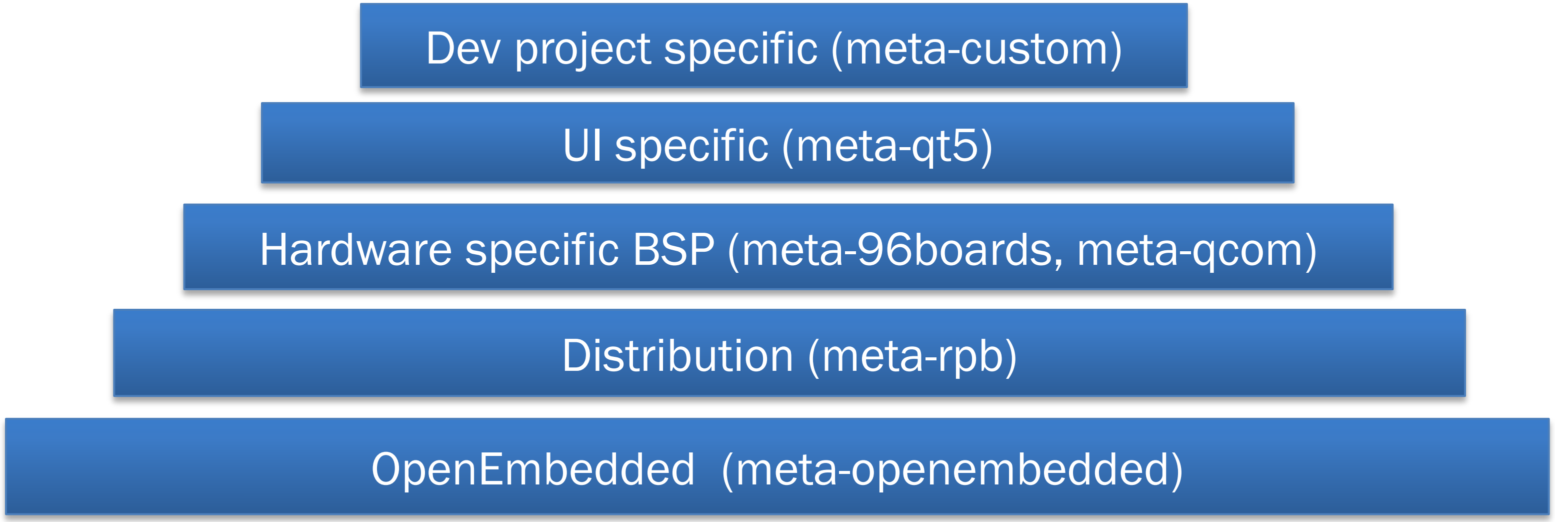
OpenEmbedded RPB — BSP customizations

- **There are many ways and processes to customize Yocto BSP**
  - Driven by the customization type and scope
  - Examples:
    - Adding a new API to OpenEmbedded RPB BSP
    - Modifying API package configuration
    - Modifying API code
    - Defining a new RFS content
    - Extending OpenEmbedded RPB BSP to support custom hardware
- **Customizations for our product**
  - Focusing on the RFS – APIs
  - Adding required software to the reference BSP
  - Process we will follow:
    1. Create a product specific metalayer
    2. Develop a recipe for an API package
    3. Extend RFS definition to include required APIs
    4. Build the customized OpenEmbedded RPB BSP/SDK

OpenEmbedded RPB customizations (1)

Dedicated Metalayer

- Metalayer is a group of files – recipes, configuration files, which provide a support for a specific feature or functionality
- Organized under metalayer directory
- Examples of existing metalayers
  - meta-qcom
  - meta-qt5
- Provides organized approach to defining new features/functionality
- Allows us to overwrite/extend functionality defined in other metalayers



Dev project specific (meta-custom)

UI specific (meta-qt5)

Hardware specific BSP (meta-96boards, meta-qcom)

Distribution (meta-rpb)

OpenEmbedded (meta-openembedded)

OpenEmbedded RPB customizations (2)

Process

- Copy one of existing metalayers and rename it i.e.

```
$ cp –r meta-backports meta-custom
```

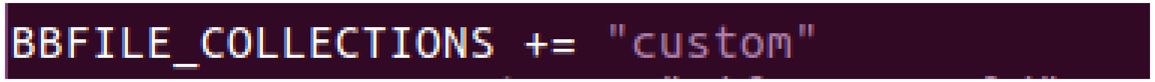
- Remove existing recipe directories

```
$ cd meta-custom
$ rm –rf browser* COPYING* core meta* openembedded* README
```

- Create your own recipes directory

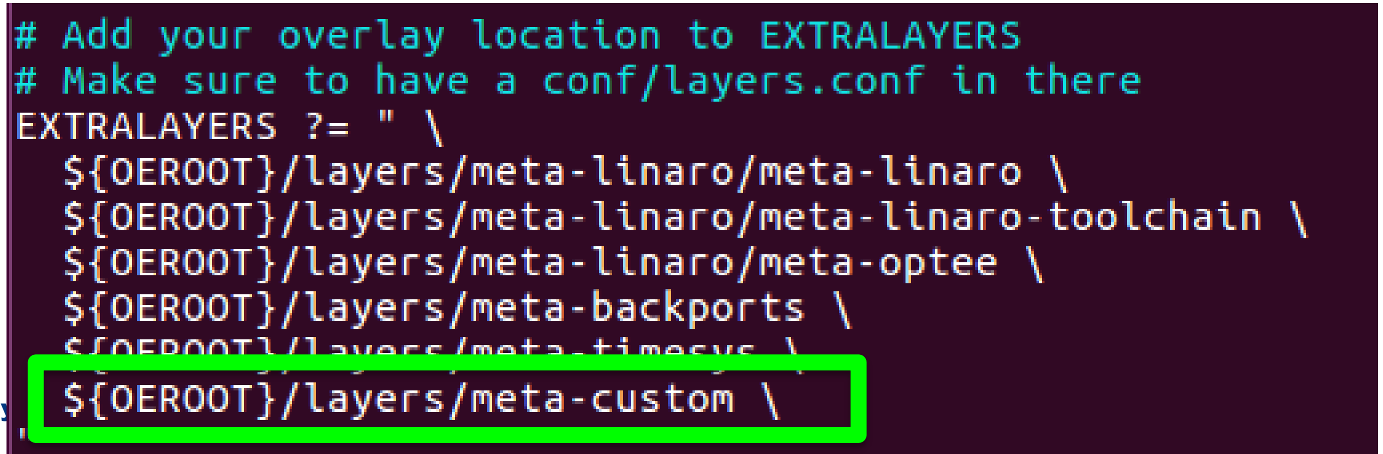
```
$ mkdir recipes-custom
```

- Change name in conf/layer.conf



```
BBFILE_COLLECTIONS += "custom"
```

- Add new metalayer to the conf/bblayers.conf file in your BSP build directory



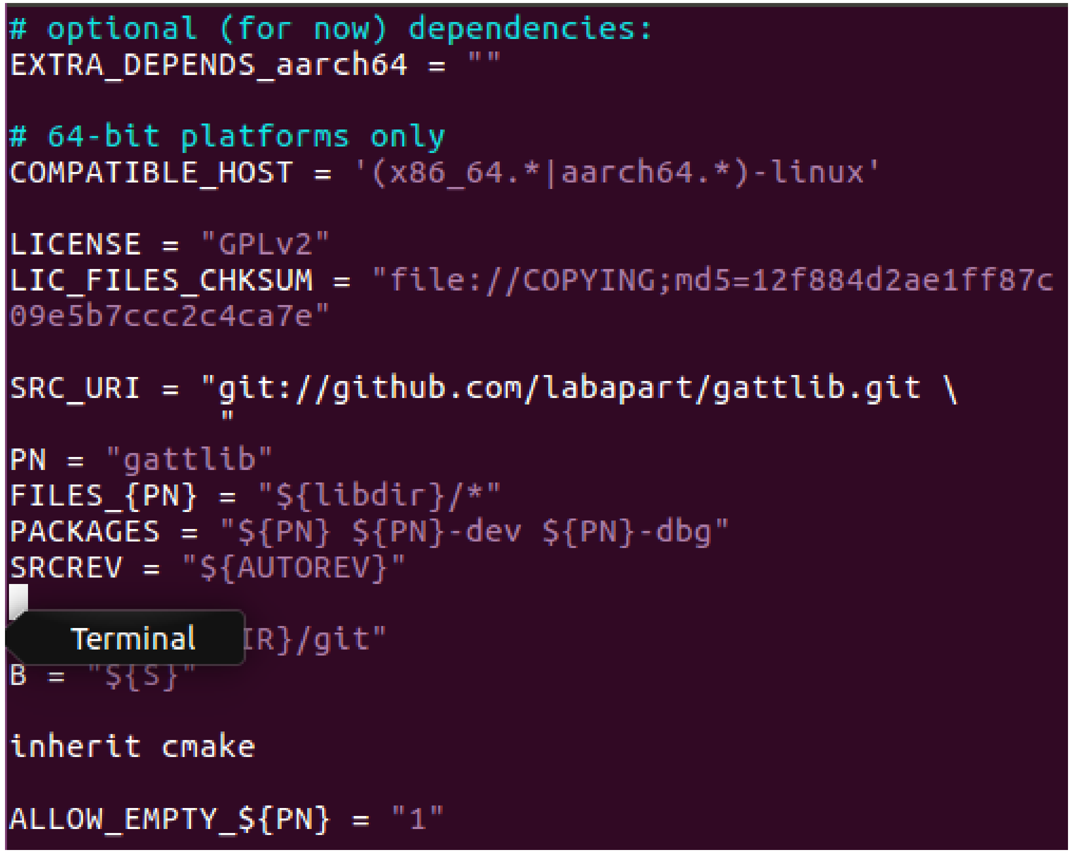
```
# Add your overlay location to EXTRALAYERS
# Make sure to have a conf/layers.conf in there
EXTRALAYERS ?= " \
  ${OEROOT}/layers/meta-linaro/meta-linaro \
  ${OEROOT}/layers/meta-linaro/meta-linaro-toolchain \
  ${OEROOT}/layers/meta-linaro/meta-optee \
  ${OEROOT}/layers/meta-backports \
  ${OEROOT}/layers/meta-timesys \
  ${OEROOT}/layers/meta-custom \
```

OpenEmbedded RPB customizations (3)

Adding a new API

- Requires a new recipe
- Can be added in the dedicated metalayer
- Create recipes-custom directory
- Inside, create a directory for every package you are adding
- Save the recipe in a package/API folder

Note: New meta-custom layer and recipes will be available to you for download after this session.



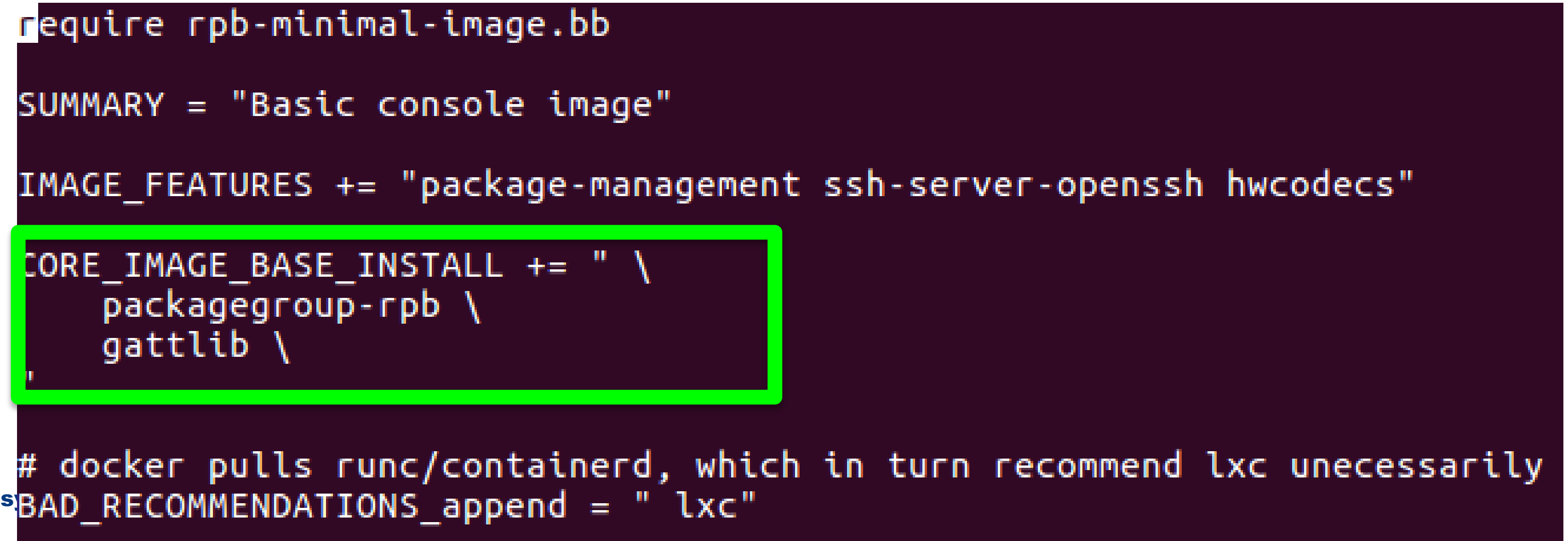
```
# optional (for now) dependencies:
EXTRA_DEPENDS_aarch64 = ""

# 64-bit platforms only
COMPATIBLE_HOST = '(x86_64.*|aarch64.*)-linux'

LICENSE = "GPLv2"
LIC_FILES_CHKSUM = "file://COPYING;md5=12f884d2ae1ff87c
09e5b7ccc2c4ca7e"

SRC_URI = "git://github.com/labapart/gattlib.git \
          "

PN = "gattlib"
FILES_{PN} = "${libdir}/*"
PACKAGES = "${PN} ${PN}-dev ${PN}-dbg"
SRCREV = "${AUTOREV}"

              IR}/git"
B = "${S}"

inherit cmake

ALLOW_EMPTY_${PN} = "1"
```

OpenEmbedded RPB customizations (4)

- **Extending BSP filesystem**
- **Process:**
  - Copy the original rpb-console-image.bb to meta-custom

```
$ cd meta-custom/recipes-custom
$ mkdir images
$ cp ../../meta-rpb/recipes-samples/images/rpb-console-image.bb \
        images/rpb-console-image-session2.bb
```

  - Copy also rpb-minimal-image.bb
  - Edit the rpb-console-image-session2.bb – add gattlib package



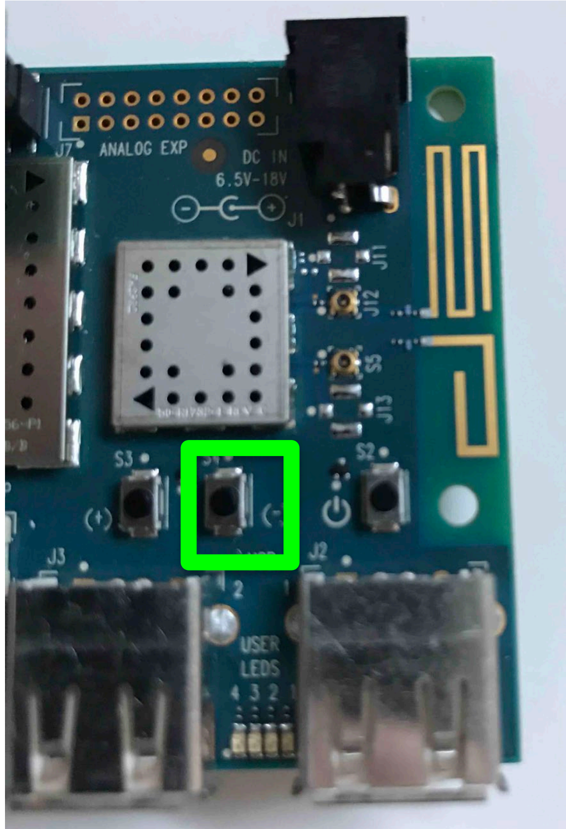
```
require rpb-minimal-image.bb

SUMMARY = "Basic console image"

IMAGE_FEATURES += "package-management ssh-server-openssh hwcodecs"

CORE_IMAGE_BASE_INSTALL += " \
    packagegroup-rpb \
    gattlib \
"

# docker pulls runc/containerd, which in turn recommend lxc unecessarily
BAD_RECOMMENDATIONS_append = " lxc"
```

OpenEmbedded RPB customizations (4)

- Build the new, custom image
- Run bitbake from the build directory

```
$ bitbake rpb-console-image-session2
$ bitbake rpb-console-image-session2 –c populate_sdk
```

- Deploy the image to the DragonBoard 410c
  - DragonBoard 410c with existing software can be put in the fastboot mode with the following procedure
    - Power down the system
    - Press and Hold S4 button (-)
    - Power on the system
    - Connect from host with fastboot and flash new images using commands learned in session 1



LAB 1 — OpenEmbedded RPB BSP customization

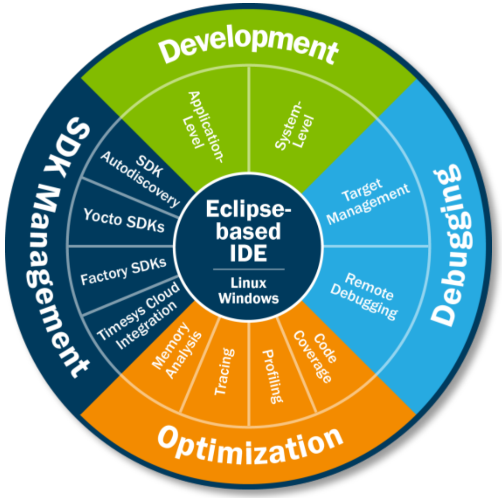
- **Create a meta-custom**
- **Add gattlib API**
  - Create needed directory structure
  - Write a gattlib.bb recipe
- **Add gattlib to bblayers**
- **Extend rpb-console-image**
  - Create a custom rpb-console-image-session2.bb based on original rpb-console-image.bb
  - Add the gattlib package to the rpb-console-image-session2.bb
- **Build the custom BSP image and the matching SDK**
- **Install the custom BSP image on the DragonBoard 410c**
- **Install the custom SDK on the Host PC**

Application Development Setup

IDE considerations

- **Integrated Development Environments help greatly with software development**
  - Keep the workspace organized
  - Promote and support collaborative development
  - Simplify and accelerate coding, debugging and software optimization
- **Eclipse™ framework — most popular IDE baseline leveraged by many tools**
  - TimeStorm™ — IDE used in this session
- **Development environment Setup**

+

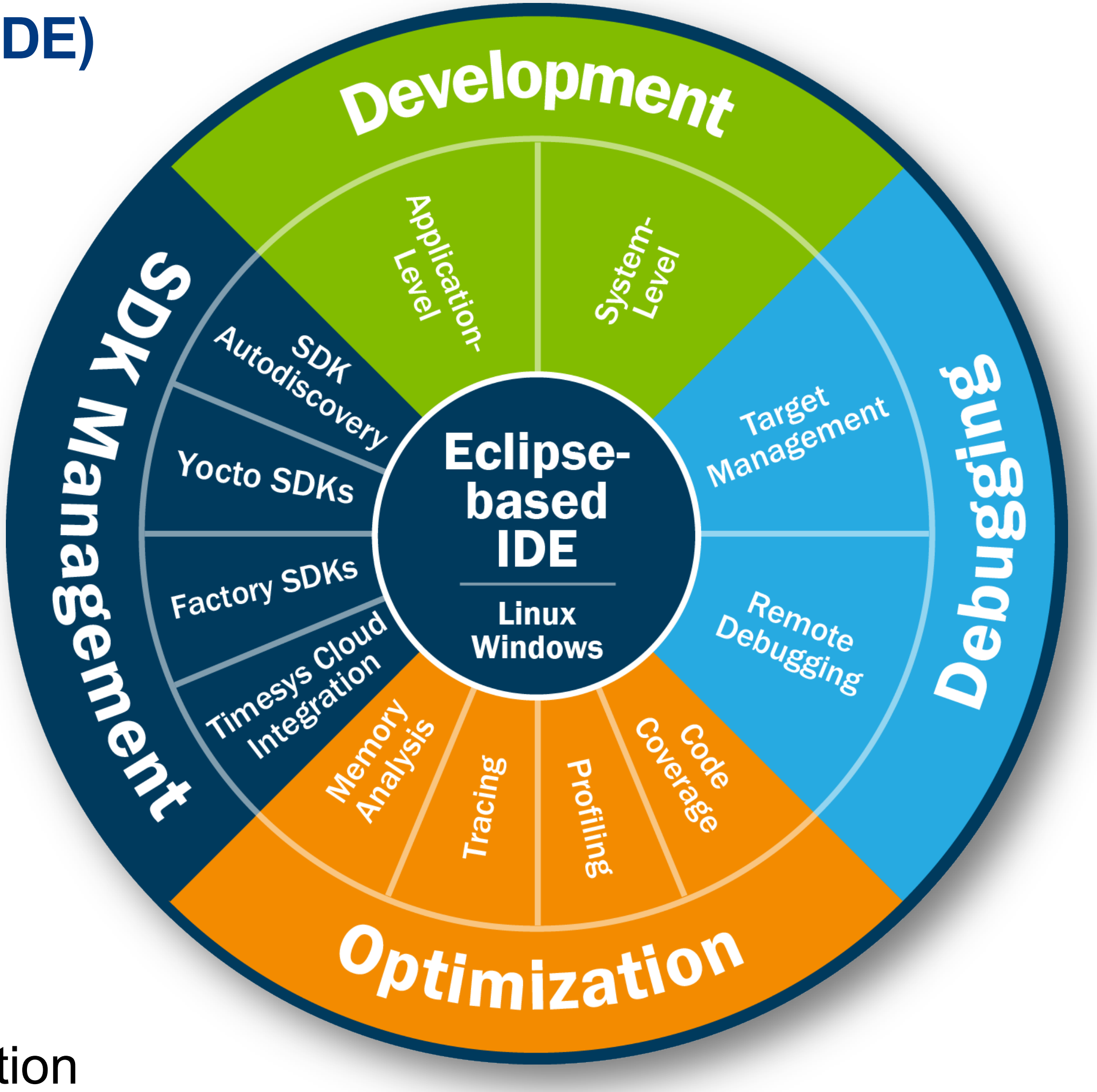


+

Java

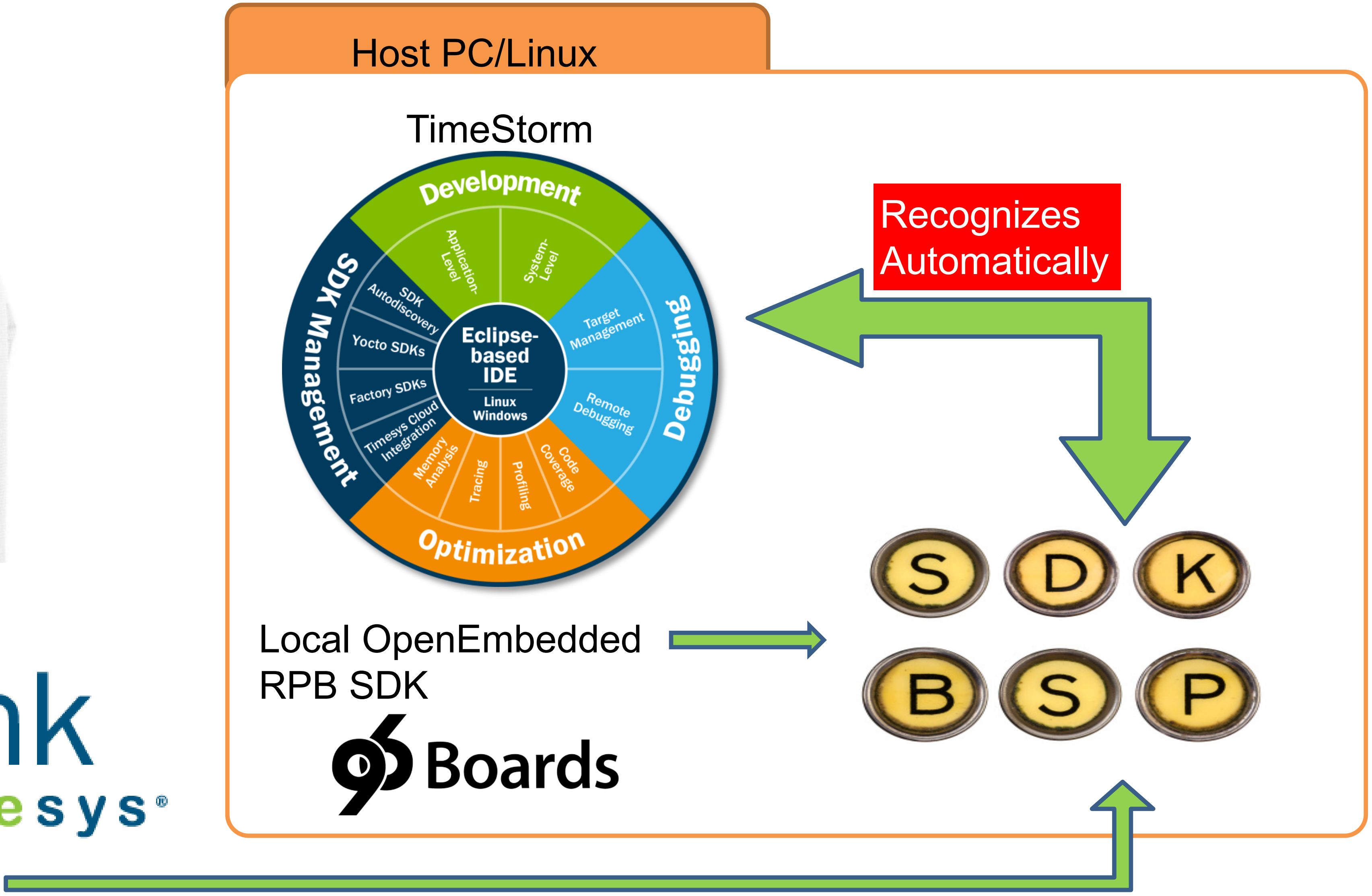
TimeStorm IDE

- **Integrated Development Environment (IDE)**
- **Extends Eclipse Project**
- **Focus:**
  - Embedded systems
  - SDK management
  - Target management
  - Application development
  - Linux kernel development
  - Device driver development
  - Debugging (kernel, application)
  - Profiling
    - Gprof
    - OProfile
  - Code coverage
  - Memory leak analysis
  - Tracing (LTTng)
  - Factory™/Yocto/OpenEmbedded integration



IDE and Cross-Environment

LinuxLink
by timesys



Host PC/Linux

TimeStorm

Recognizes Automatically

Local OpenEmbedded
RPB SDK

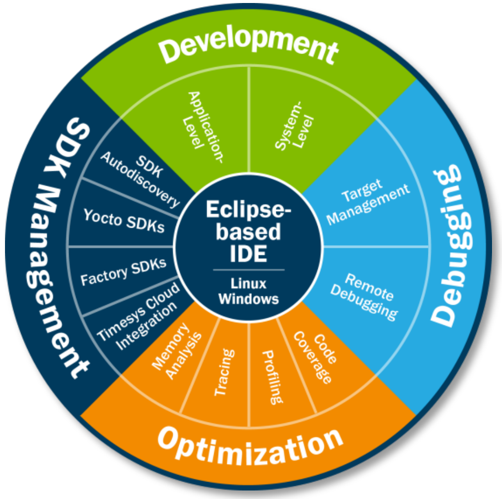
96Boards

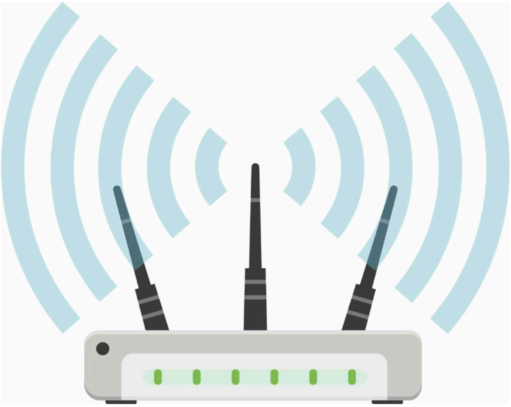
Application Development

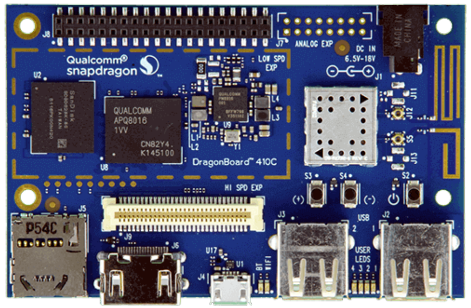
Application Development Environment Setup

- **Download TimeStorm (http://linuxlink.timesys.com)**
  - Free registration required
  - You get free 30-day license for the fully featured IDE
- **Download and install Oracle® Java ® JRE version 8 (http://www.oracle.com/technetwork/java/javase/downloads/index.html)**
- **Make sure both your DragonBoard 410c and your Host PC have IP address assigned**

Host PC







Connecting DragonBoard 410c via TCP/IP

- **DragonBoard 410c comes equipped with the WiFi chipset**
- **Firmware is available and preinstalled in the custom RPB BSP we built**
- **Procedure to connect to a WiFi router:**

1. Show all connections

```
$ nmcli connection show
```

1. Show device status

```
$ nmcli device status
```

2. View the list of available access points

```
$ nmcli dev wifi list
```

3. Create a connection

```
$ nmcli con add con-name WiFi ifname wlan0 type wifi ssid foonet
```

    1. Setup password (if needed)

```
$ nmcli con modify WiFi wifi-sec.key-mgmt wpa-psk
$ nmcli con modify WiFi wifi-sec.psk mypassword
```

    2. Enable the connection

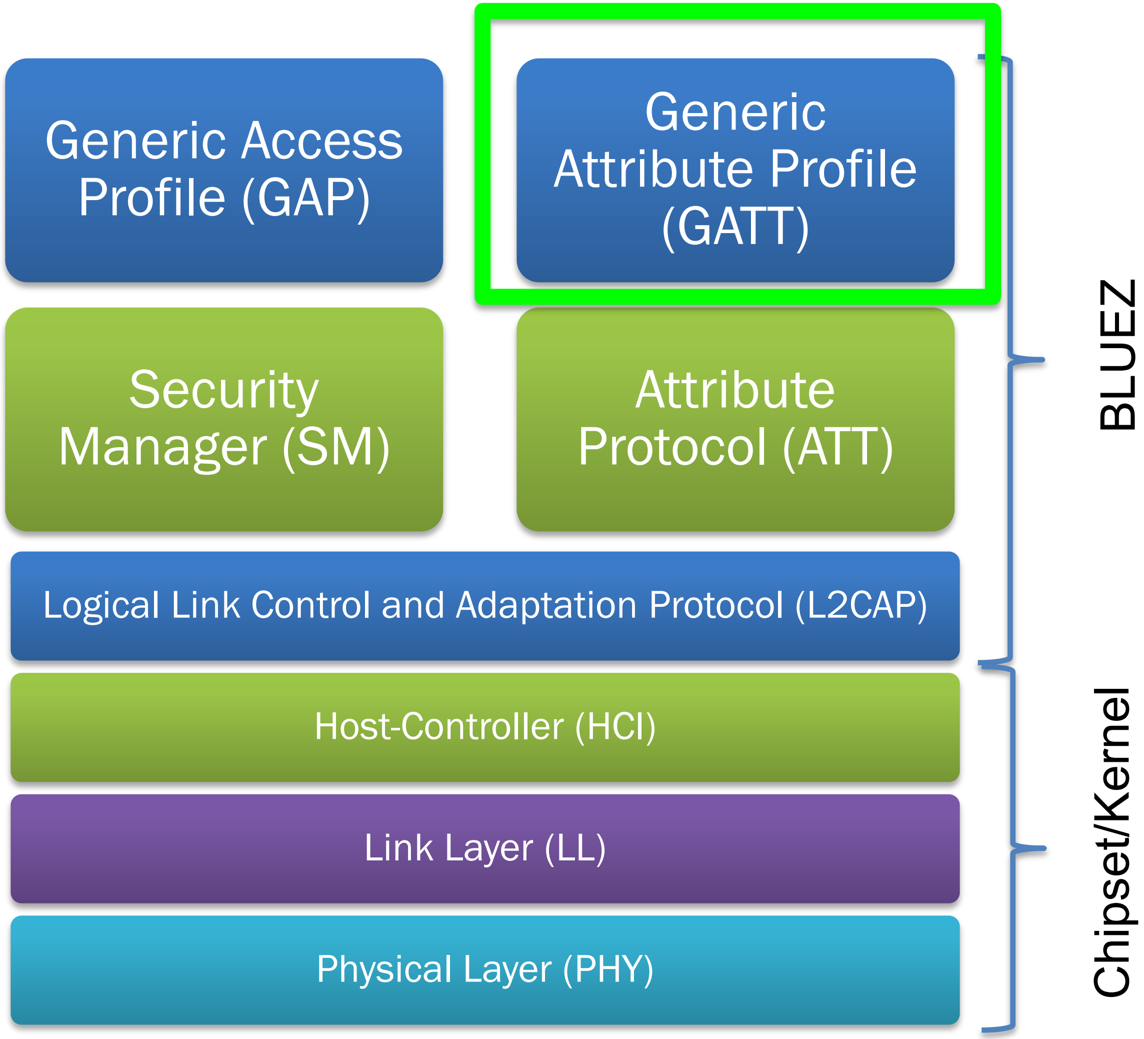
```
$ nmcli con up WiFi
```

Gathering Sensor Data

- **Linux BSP deployed on DragonBoard 410c has Bluetooth enabled**
- **SensiBLE (http://www.sensiedge.com)**
  - Collects data from onboard sensors
  - Makes them available via BLE protocol
- **We need to:**
  - Establish BLE connection
  - Receive sensor BLE messages
  - Process the information — analyze and trigger an action

Application Development (1) – BLE Protocol

- **GATT – Generic Attribute Profile**
  - Defines a way two BLE devices exchange data
  - Use concepts of
    - Services
    - Characteristics
  - Requires that connection is already advertised – managed by GAP
  - GATT Connections are one-to-one (exclusive)
    - Once a BLE peripheral connects to DragonBoard 410c, it will stop advertising itself
  - In Linux we can manage communications with
    - gatttool (utility)
    - gattlib (programming API)



LAB 2 – Setup Application Development Environment

- **Start the IDE**
  - Verify that the newly installed SDK has been discovered
    - Navigate in the IDE to:

      Window > Preferences > TimeStorm > SDKs > Yocto SDKs
- **Connect DragonBoard 410c to the wireless router, obtain an IP address**
- **Verify connection to the target**
  - Use ping command from host to the target
  - Use IDEs Target Management
    - Setup the DragonBoard 410c — Use ssh and scp for connection
    - Verify connection with “Check Link” button

Application Development (2)

- **We will develop C application called IIoT Concentrator**
- **Available advertised characteristics**

Source: http://www.sensiedge.com/

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Feature	RFU	ADPCM Sync	Switch	Direction of arrival	ADPC Audio	MicLevel	Proximity	Lux	Acc	Gyro	Mag	Pressure	Humidity	Temperature	Battery	Second Temperature

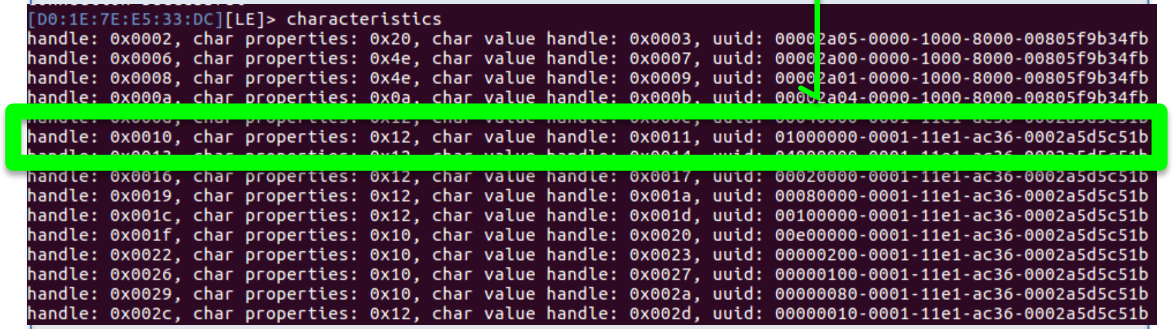
Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Feature	RFU	RFU	RFU	RFU	Beam forming	Acc Event	Free Fall	Sensor Fusion	Sensor Fusion	Compass	Motion intensity	Activity	Carry Position	Proximity Gesture	Mems Gesture	Pedometer

- **The application shall:**
  - Scan the hci0 interface
  - Discover BLE devices
  - Discover supported characteristics
  - Setup notifications on select characteristics
    - Temperature
    - Light
    - Free fall
  - Run analytics based on received data
  - Display on the console
    - Warning message – not enough light
    - Alert – Temperature too high
    - Info – Robotic arm moved fast down

- **Characteristics represented by a 32 bit value**

$$\underbrace{0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}_{\text{0x1000000}}$$

- **Advertised characteristic: LUX**



```
[D0:1E:7E:E5:33:DC][LE]> characteristics
handle: 0x0002, char properties: 0x20, char value handle: 0x0003, uuid: 00002a05-0000-1000-8000-00805f9b34fb
handle: 0x0006, char properties: 0x4e, char value handle: 0x0007, uuid: 00002a00-0000-1000-8000-00805f9b34fb
handle: 0x0008, char properties: 0x4e, char value handle: 0x0009, uuid: 00002a01-0000-1000-8000-00805f9b34fb
handle: 0x000a, char properties: 0x0a, char value handle: 0x000b, uuid: 00002a04-0000-1000-8000-00805f9b34fb
handle: 0x000d, char properties: 0x12, char value handle: 0x000e, uuid: 00040000-0001-11e1-ac36-0002a5d5c51b
handle: 0x0010, char properties: 0x12, char value handle: 0x0011, uuid: 01000000-0001-11e1-ac36-0002a5d5c51b
handle: 0x0013, char properties: 0x12, char value handle: 0x0014, uuid: 04000000-0001-11e1-ac36-0002a5d5c51b
handle: 0x0016, char properties: 0x12, char value handle: 0x0017, uuid: 00020000-0001-11e1-ac36-0002a5d5c51b
handle: 0x0019, char properties: 0x12, char value handle: 0x001a, uuid: 00080000-0001-11e1-ac36-0002a5d5c51b
handle: 0x001c, char properties: 0x12, char value handle: 0x001d, uuid: 00100000-0001-11e1-ac36-0002a5d5c51b
handle: 0x001f, char properties: 0x10, char value handle: 0x0020, uuid: 00e00000-0001-11e1-ac36-0002a5d5c51b
handle: 0x0022, char properties: 0x10, char value handle: 0x0023, uuid: 00000200-0001-11e1-ac36-0002a5d5c51b
handle: 0x0026, char properties: 0x10, char value handle: 0x0027, uuid: 00000100-0001-11e1-ac36-0002a5d5c51b
handle: 0x0029, char properties: 0x10, char value handle: 0x002a, uuid: 00000080-0001-11e1-ac36-0002a5d5c51b
handle: 0x002c, char properties: 0x12, char value handle: 0x002d, uuid: 00000010-0001-11e1-ac36-0002a5d5c51b
```

LAB 3 — C Application Development

- **IIoT Concentrator application source code review**
- **Compile and deploy the application in TimeStorm**
- **Deploy and run the code on the DragonBoard 410c**
- **Debug application remotely on the DragonBoard 410c**
- **Advanced analysis of the developed code**
  - Verify code coverage
  - Profile application through instrumentation
  - Perform memory analysis searching for memory leaks
  - Trace the application execution in time

Questions?

developer.qualcomm.com 96boards.org arrow.com timesys.com

Thank you

Follow us on:

For more information, visit us at:
www.qualcomm.com & www.qualcomm.com/blog

All data and information contained in or disclosed by this document is confidential and proprietary information of Qualcomm Technologies, Inc. and/or its affiliated companies and all rights therein are expressly reserved. By accepting this material the recipient agrees that this material and the information contained therein is to be held in confidence and in trust and will not be used, copied, reproduced in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies, Inc. Nothing in these materials is an offer to sell any of the components or devices referenced herein.

Qualcomm and DragonBoard are trademarks of Qualcomm Incorporated, registered in the United States and other countries. Other products and brand names may be trademarks or registered trademarks of their respective owners.

References in this presentation to “Qualcomm” may mean Qualcomm Incorporated, Qualcomm Technologies, Inc., and/or other subsidiaries or business units within the Qualcomm corporate structure, as applicable. Qualcomm Incorporated includes Qualcomm’s licensing business, QTL, and the vast majority of its patent portfolio. Qualcomm Technologies, Inc., a wholly-owned subsidiary of Qualcomm Incorporated, operates, along with its subsidiaries, substantially all of Qualcomm’s engineering, research and development functions, and substantially all of its product and services businesses, including its semiconductor business, QCT.