Embedded Development Made Easy

A Guide to Open Source Platforms and Maximizing value for your product

Bassem Nomany
bassem.nomany@exodo.engineering

About me

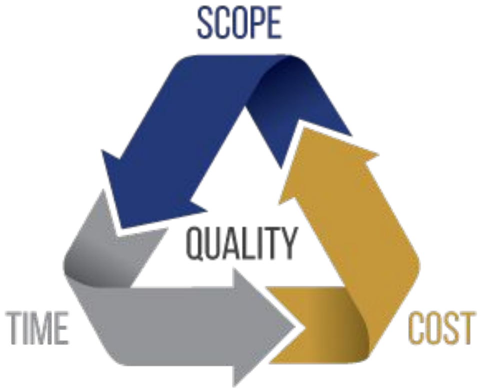
- System architect with 13 years of experience in embedded systems, IoT, and software-defined products.
- Worked across industries: automotive infotainment, electric mobility and smart connected devices.
- Worked extensively with open source and commercial embedded platforms.
- Freelancer working to help OEMs develop secure, software-defined products with open source platforms at the core.

What this talk is about

- Engineering is balancing act to trade-off time to deliver features, lowest cost, and best quality.
- Modern software is rich in features and comes with high expectations.
- Embedded platforms act as backbone to integrate and manage, contain or encapsulate complexity.

This presentation is high level guide for:

- Makers who want to launch a product.
- Startups trying to get started.
- OEMs buried in complexity.
- Anyone feeling stuck between vendor lock-in.

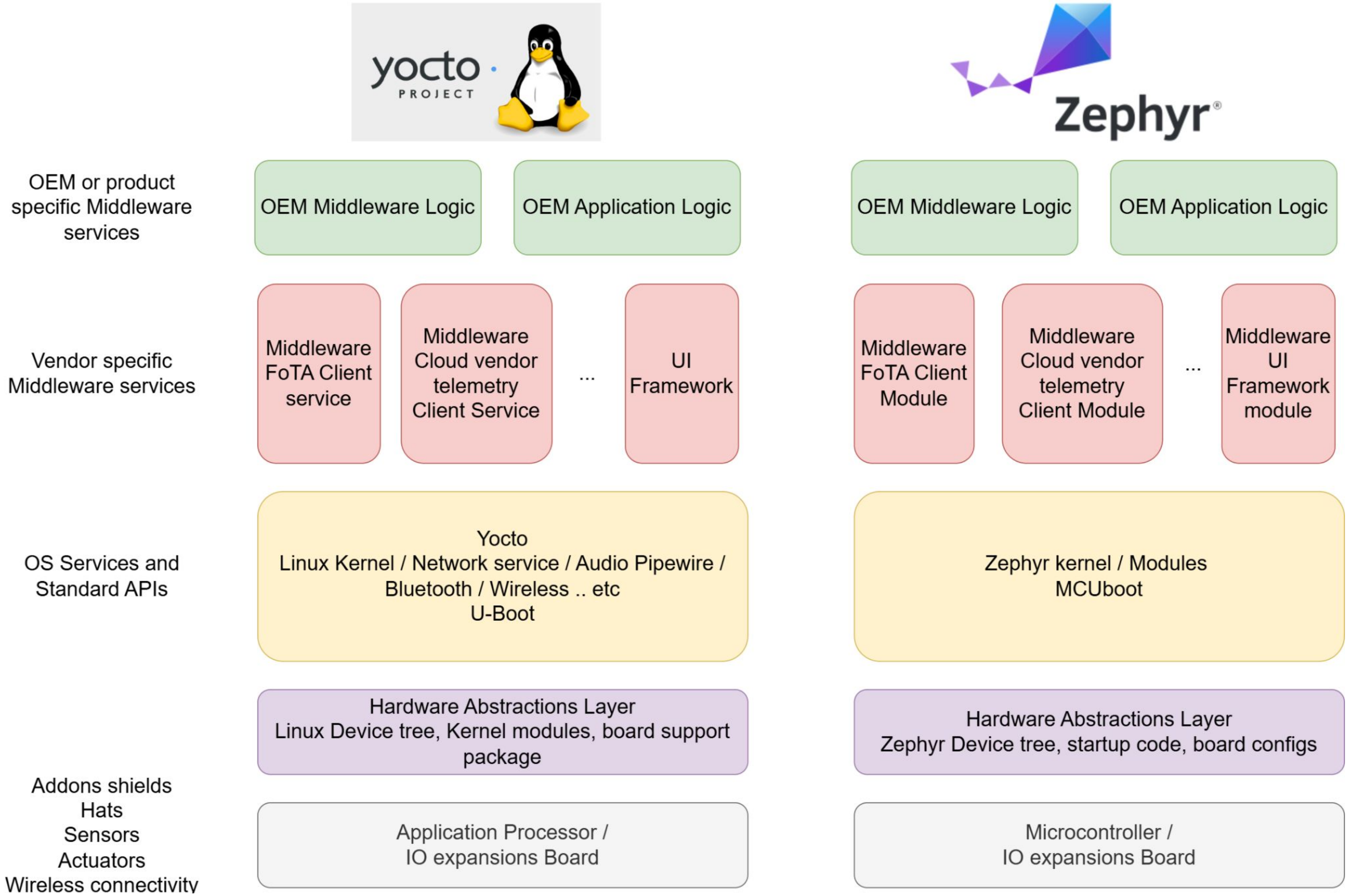


RTOS or full-fledged operating system?

Application Processor or MCU?

What’s the best embedded ecosystem that FOSS can offer my project?

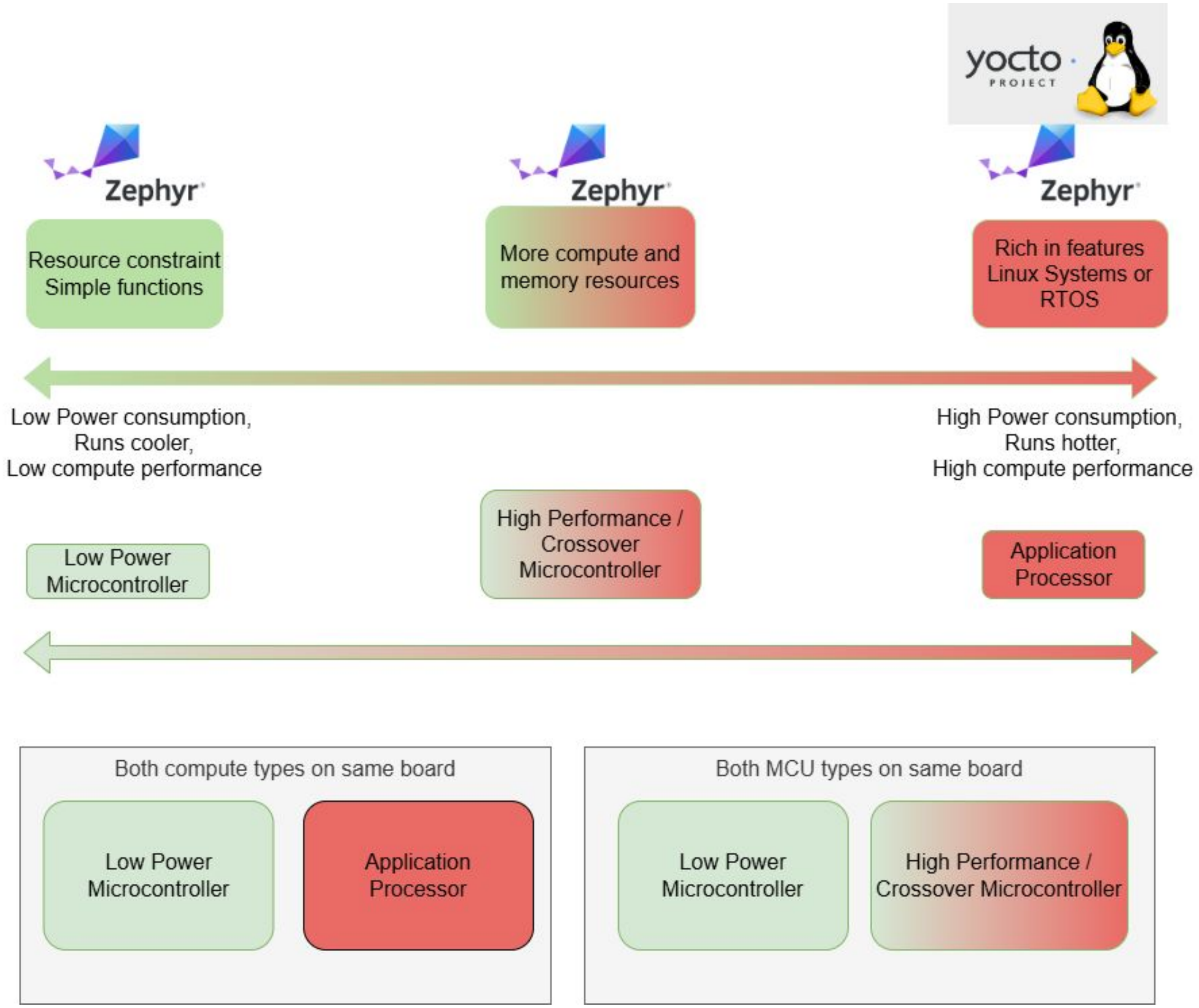
That’s today’s topic …


yocto PROJECT
Zephyr
OEM or product specific Middleware services
OEM Middleware Logic
OEM Application Logic
OEM Middleware Logic
OEM Application Logic
Vendor specific Middleware services
Middleware FoTA Client service
Middleware Cloud vendor telemetry Client Service
...
UI Framework
Middleware FoTA Client Module
Middleware Cloud vendor telemetry Client Module
...
Middleware UI Framework module
OS Services and Standard APIs
Yocto
Linux Kernel / Network service / Audio Pipewire / Bluetooth / Wireless .. etc
U-Boot
Zephyr kernel / Modules
MCUboot
Hardware Abstractions Layer
Linux Device tree, Kernel modules, board support package
Hardware Abstractions Layer
Zephyr Device tree, startup code, board configs
Addons shields
Hats
Sensors
Actuators
Wireless connectivity
Application Processor / IO expansions Board
Microcontroller / IO expansions Board

Both platforms offer:

- Layered architecture, and standard services APIs.
- Hardware abstractions that facilitates swapping hardware or building for different hardware variants.
- A hardware variant can be different product hardware generation, or different market segment hardware, or simulation.
- Wide variety of peripherals and boards already offer reference integration.
- Portability: Write once, deploy on multiple devices with minimal changes
- Unified sensor and driver integration via standard interfaces
- Cover a range of functionalities: Basic I/O, sensor interfacing, networking, etc.

FOSS platforms scales across devices types


yocto PROJECT
Zephyr
Zephyr
Zephyr
Resource constraint Simple functions
More compute and memory resources
Rich in features Linux Systems or RTOS
Low Power consumption, Runs cooler, Low compute performance
High Power consumption, Runs hotter, High compute performance
Low Power Microcontroller
High Performance / Crossover Microcontroller
Application Processor
Both compute types on same board
Low Power Microcontroller
Application Processor
Both MCU types on same board
Low Power Microcontroller
High Performance / Crossover Microcontroller

FOSS platforms scales across devices types

Traditionally, the split between MCU world and APU world has been clear cut.
Nowadays, Compute offering is more of a spectrum where you try to place your product on the spectrum based on requirements and specifications.
One can over spec, or under spec depending on the use cases and maturity of requirements.

High performance MCUs or Crossovers MCUs can offer acceleration for:

- 2.5D Graphics
- Camera Interface
- Multi-channel I2S audio

There’s boards that can offer both compute types and run different compute based on the use case to get benefits of both worlds.

Typical case of each compute type:

- Low Power example: Battery powered devices, smart tags, Low power wireless communication devices for IoT can run software based rendering too.
- Crossovers MCUs: Two wheelers, like Motorcycles, Instrument clusters for cars, Smart home with displays
- APUs: Can be found in complex functions like infotainment, Home Theaters, Application based, Navigation systems.
- One can run even Zephyr or Linux, or Zephyr and Linux sametime using Hypervisor on APUs

Yocto - Zephyr similarities and differences (1 / 3)

- Yocto Linux is build system used to create custom Linux distributions for embedded systems on **more powerful processors**.
- Example: MPUs (ARM Cortex-A, x86, etc.)
- Smallest unit of software are contained in recipes.
- A collection of logically grouped recipes are contained in a layer.
- A manifest specifies which layers to fetch to build the distro.
- Uses **repo tool** to fetch the layers.
- Uses **bitbake** to build and test images.

- Zephyr RTOS focus on **resource constrained** devices, with tighter requirements around real time.
- Example: MCUs (ARM Cortex-M)
- Buildable software is organized into modules which have its own git repository.
- Relies on a manifest to fetch modules into sources directory
- Uses **west** tool to fetch sources for all the modules
- Uses **west** tool to build, flash, debug and test images

Yocto - Zephyr similarities and differences (2 / 3)

Tailor and Configure

- Both platforms are highly customizable in terms of:
  - Sources to fetch from remote repositories.
  - Selecting out of the box or custom boards BSP.
  - Both are open-source projects with strong community and corporate backing.
  - Actively maintained with frequent releases and broad contributor bases.
  - Configured in similar ways:
    - Devicetree for describing hardware nodes and peripherals attached or living on the board
    - Kconfig (guiconfig)to describe what modules to include in the zephyr image, or Linux Kernel

Yocto - Zephyr similarities and differences (3 / 3)

Runtime

- Linux organizes **logical** software **functions** in **various processes**, which runs in **separate memory spaces**.
- Relies on **Systemd as init process** to start the image processes units in defined sequence called boot targets.
- A **crashing userspace process** would mean high chance of **recovery** if systemd was instructed to restart on failure.
- Linux Rootfs **boot time** to full function can take **seconds**.

- Zephyr RTOS organizes **logical** software **functions** in different **threads**, which runs in **one monolith process**.
- **Threads can be created and started** from main function or **K_THREAD_ macros**.
- A **crashing thread** would mean **crashing entire OS** and its application.
- Zephyr RTOS have impressive **boot time**, typical hundred of **milliseconds**.

Yocto - Zephyr similarities and differences (3 / 3)

The difference is between the Operating systems is how the features and complexity is managed.

- **Linux** is a full-fledged general-purpose operating system with:
  - Memory management (MMU support)
  - User space and kernel space separation
  - Device drivers for a wide range of hardware
  - Multi-user and multi-process support
  - Filesystems, networking, security, etc.
- **RTOS** is minimal by design:
  - Tailored for specific hardware and use case
  - Focus on minimal necessary components (e.g., task scheduler, interrupt handling)
  - No user space/kernel space separation (monolithic)

Linux sophisticated architecture Impact: More code, more processes and more initialization steps in Linux = longer boot time.

Modern expectations (1 / 7) - Software-defined products

- Manufacturers figured out a business model to subsidize the investment and spread it over long period of time.
- Instead of allocating big capital for the product, they can sell MVP and use the profits to fund the next set of feature to roll out and so on.
- Project capital investment is broken down on much longer time after product launch.
- This allows for:
    - Lower the risk of capital investment without return on investment.
    - Prove product demand and validate the market growth before investing in full blown product with features users might not need or require.
    - Products today are created in phases and have a feature rollout plans based on core features, and extra features.
    - Consumers can vote with their wallet and adoption

Modern expectations (1 / 7) - Software-defined products

- Launch MVP before building the whole solution
- Rollout, test features, monitor and adapt to user feedback and response
- Adapt to ever changing regulations and market competition landscape

- Decouple Software features and behavior from hardware functions
- A need for scalable platform for feature growth rollouts, easy to change and evolve
- Over the air updates to update the device behavior and functions

- **Decoupling** is done via **HAL** using **BSP layers**
- Offer service oriented architecture using IPCs like **dbus**
- Offer Software packaging (rpm/deb) for dependencies organization
- Offer FoTA solution with great integration for both apps and full system image using **u-boot** and **pkg manager**.

- **Decoupling** is done via **HAL** using **board selection during west build**
- Offer service oriented architecture using threads bus communication (**zbus**)
- Can offer Linkable Loadable Extensions (LLM) for adding/updating features without rebooting.
- Offer FoTA solution with great integration for full system image using **MCUboot**.

Modern expectations (2 / 7) - Continuous security & regulations

With EU Cyber Resilience Act (CRA) and RED (Radio Equipment Directive) updates, the game is changing. Products must now:

- Stay secure throughout their entire lifecycle, and offer faster security response.

- Bitbake can generate SBOMs in SPDX or CycloneDX format using meta-security.
- meta-security layer enables CVE tracking, vulnerability scanning, and integration with:
    - CVE database monitoring
    - cve-check.bbclass to flag known issues
    - Tools like cve-bin-tool

- Zephyr can generate SPDX SBOMs automatically using west build tools.
- The Zephyr project itself tracks CVEs.
- Security patches are regularly backported to LTS releases.

Modern expectations (2 / 7) - Continuous security & regulations

- One-time reviews is not enough, vulnerabilities are typically found after software has been shipped.
- SBOMs detail modules’ or components’ versions used
- CVE scanners and checkers uses SBoMs to create updated CVEs list and generate reports along with known fixes.
- Your project dependency tree must be visible and traceable.
- Both Zephyr and Yocto projects can generate SBOMs as build artifact and maintain them over time.

Both platforms offer security Guidelines for creating secure solutions:

- Secure boot via u-boot, or MCUboot, Secure storage.
- Protecting data at rest, and Data in transit.

Modern expectations (3 / 7) - Testing

yocto PROJECT

- **Automated testing via BitBake**: oe-selftest, ptest for functional and regression testing
- So many unit testing frameworks based on language and technology, such as Google Test (gtest/gmock)
- Emulation testing via **QEMU**
- **OTA testing integration** with Mender, SWUpdate
- Hardware-in-the-Loop (HIL) using CI runners on Jenkins or Gitlab Runners.

Zephyr

- **Automated testing via west**: Twister Test Runner for Unit, integration, driver, and platform tests.
- Offers Ztest framework and Fake Function Framework for unit and mocking testing.
- Emulation testing via **QEMU & native_sim**
- **OTA testing integration** with Mender (Preview)
- Hardware-in-the-Loop (HIL) using CI runners on Jenkins or Gitlab Runners.

Modern expectations (4 / 7) - CI/CD

There's many offerings for Continuous Integration and deployment that can easily setup with **Jenkins pipeline**, **Gitlab CI**, **Github actions**.

Build Instrumentation Wraps around **BitBake (Yocto)** or **West (Zephyr)** with manifests for reproducible builds.

Simulation Testing: Run QEMU or native_sim in CI for early validation

Hardware Testing at Scale: Connected device farm for automated HIL regression

Modern expectations (4 / 7) - CI/CD

Automated CI/CD build can automate:

- SBoMs report
- CVE scanner tools and reports
- Sign the images
- Generate test reports
- Run simulation on a target
- Run real hardware in loop on a device farm of different hardware variants.
- Store the binaries and all artifacts in artifactory

Jenkins GitLab GitHub JFrog ARTIFACTORY

Modern expectations (5 / 7) - IoT Connectivity

Lightweight M2M is more than just protocol, since it also offers a data model register per device type, and Standard data and methods for handling:

- Device onboarding and configuration
- Commissioning/decommissioning
- Software updates
- Remote factory reset
- Collecting data logs and core dump files
- More efficient in terms of data and power consumption on resource constrained

MQTT is widely adopted more than LWM2M and from cloud vendor compared to LWM2M protocol.

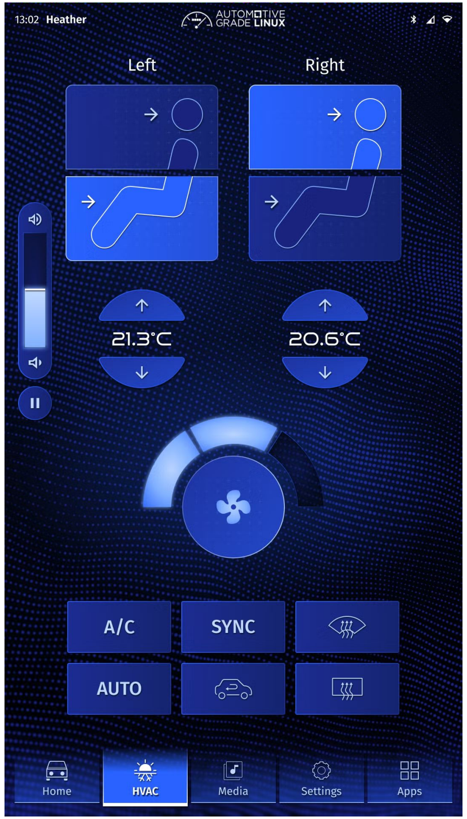
Modern expectations (5 / 7) - IoT Connectivity

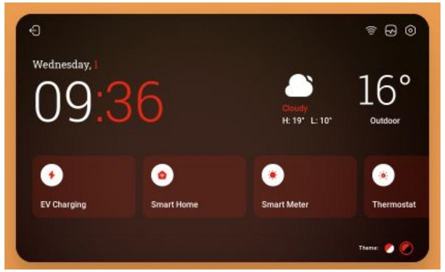
	LightweightM2M specified at oma	MQTT
Device Onboarding & Configuration	Built-in via LwM2M bootstrap & security objects	Requires custom implementation — provisioning must be built separately.
Commission/Decommission Devices	LwM2M supports secure factory reset, bootstrap & deregister flows	No standard mechanism — needs custom logic for lifecycle management
Data Telemetry & Remote Control	Read/Write/Observe model, supports CoAP Notify	Publish/subscribe model, widely adopted
Remote Monitoring & Diagnostics	Native object model for error codes, logs, crash dumps.	Requires you to define and transmit diagnostic topics manually.
Cloud Ecosystem Compatibility	Cloud agnostic services: Eclipse Leshan, EMQX	AWS IoT Core, Google Cloud IoT Core, Azure IoT hub
Security	DTLS (CoAP over UDP/TCP), supports PSK, RPK, or X.509	TLS often used, but security design is completely up to developer
Power & Network Efficiency	Designed for constrained devices; supports CoAP observe, block-wise transfers, lifetime/delay intervals.	MQTT can be chatty unless optimized (keep-alive tuning, retain flags, QoS, etc.)

Modern expectations (6 / 7) - Delightful user interfaces

Flutter

- Frameworks Supports touch, animation, themes, and fonts:
  - **Zephyr** RTOS: LVGL is **already integrated** into the Zephyr project and samples are available.
  - **Linux**: **Flutter/Dart** or **Qt/JS/C++** are great options for embedded, reference integration is available via **meta-qt6** and **meta-flutter**
- Display interfaces:
  - HDMI, MIPI-DSI, LVDS, RGB Parallel connector, SPI
  - USB or I2C for touch input data





Modern expectations (7 / 7) - ML on edge devices

Why Edge ML?

- Real-time insights without cloud dependency
- Lower latency, improved privacy, reduced bandwidth
- Enables smarter products: from wearables to robots

Modern expectations (7 / 7) - ML on edge devices

- **Frameworks:**
  - TensorFlow
  - ONNX Runtime
  - PyTorch
- Object detection (YOLO, MobileNet-SSD)
- **Complex models** depending on the processing power and GPU or NPU acceleration.

- **Framework:** TensorFlow Lite for Microcontrollers (TFLM)
- Runs in < 100KB RAM (no heap allocation)
- Simple model types:
  - Gesture recognition from accelerometer
  - Keyword spotting / sound classification
  - Vibration & sensor anomaly detection

Challenges and consideration

What questions should product creators ask before committing to Silicon footprint, vendor and choosing between Linux or RTOS based platforms?

Challenges and consideration (1 / 4) - Product requirements balancing act

What to consider when selecting MCU, APU or off the shelf board:

- Power consumption, Battery constrained?
- Compute required
- Thermal headroom or solution
- Memory required
- Peripherals availability
- Environmental rating
- Cost (BoM/SBoM)

- How many years will the device be in the market?
- What's the future features forecast for how much footprint? Should you over spec your silicon in terms of compute and memory, so it can be future ready? Or should you save on cost for affordable market segment?

Challenges and consideration (2 / 4) - Product lifecycle

- Planning product launch is exciting, **Planning** product **end of life** is **important** too.
- Long-term ownership isn’t just a business advantage. it’s becoming a legal necessity. Open source gives you the tools and freedom to own your compliance roadmap, instead of outsourcing it to someone else’s timeline.

That translates to, before selecting a MCU, APU or a board, One has to question:

- **LTS commitment** from Silicon vendor and OS platform, for how many years? 10-15 years?
- Update release **cadence commitment**? What’s the update window between new upstream release and SV updated BSP?
- What’s the **CVE response time window** for BSP?
- If SV doesn’t want to offer such LTS support, do they have binary blobs in their BSP, are they willing to **share sources for blobs**?

Challenges and consideration (3 / 4) - Hardware delays

Custom hardware development often involves lead time delays, bringup effort, driver integration, and testing in real environments.

- To kickstart the development early: Simulation, and similar EVKs can be used until real production hardware is available.
- Both Bitbake and West support building for different board variants including the production-ready board when it arrives.

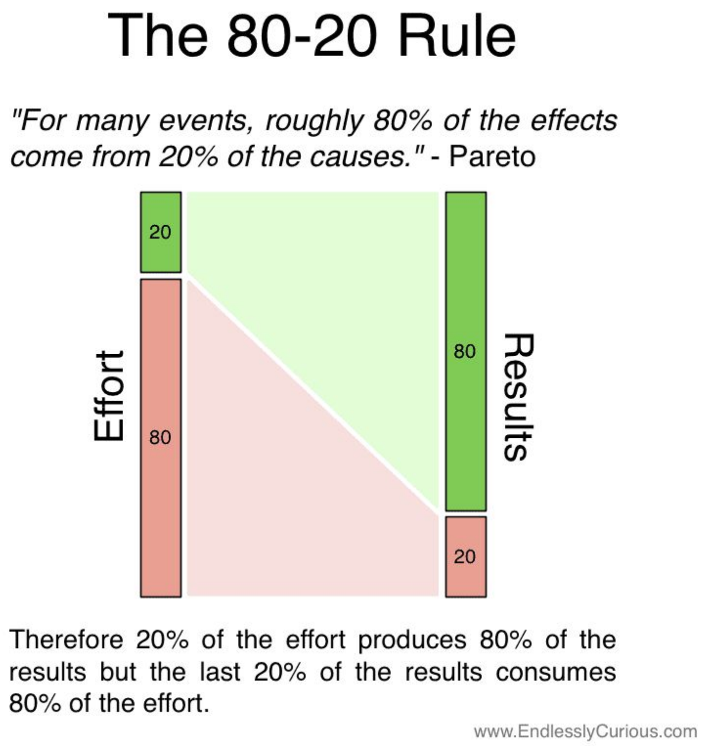
Board bring up will be less effort if the chosen Silicon vendors offers a reference integration in Zephyr for device trees and driver integration, or Yocto reference image for their APU.

Challenges and consideration (4 / 4) - Cost of ownership

80% of a platform functionality is non-differentiating (OS Kernel, Standard middleware services), 20% is your competitive edge (Your application business logic that adds value to user).

- Yocto and Zephyr platforms lets you own the 20%, while reusing the 80% for free.

Long-Term vs short-term economic Incentives for using FOSS vs Commercial



The 80-20 Rule

"For many events, roughly 80% of the effects come from 20% of the causes." - Pareto

Therefore 20% of the effort produces 80% of the results but the last 20% of the results consumes 80% of the effort.

www.EndlesslyCurious.com

Challenges and consideration (4 / 4) - Cost of ownership

Long-Term vs short-term economic Incentives for using FOSS

- Greater investment up-front, lower long-term TCO
- No per-device royalties (big deal at scale)
- Requires skilled team or partners
- Higher effort for compliance with CRA/RED
- Transparent SBOMs, auditability, and easier patchability
- Avoid vendor lock-in, and have full control over the stack (hardware to cloud)

Short-Term, commercial solution might consume less cash flow and lower upfront costs, which important to startup OEMs.

Long-Term, commercial solution is more expensive..

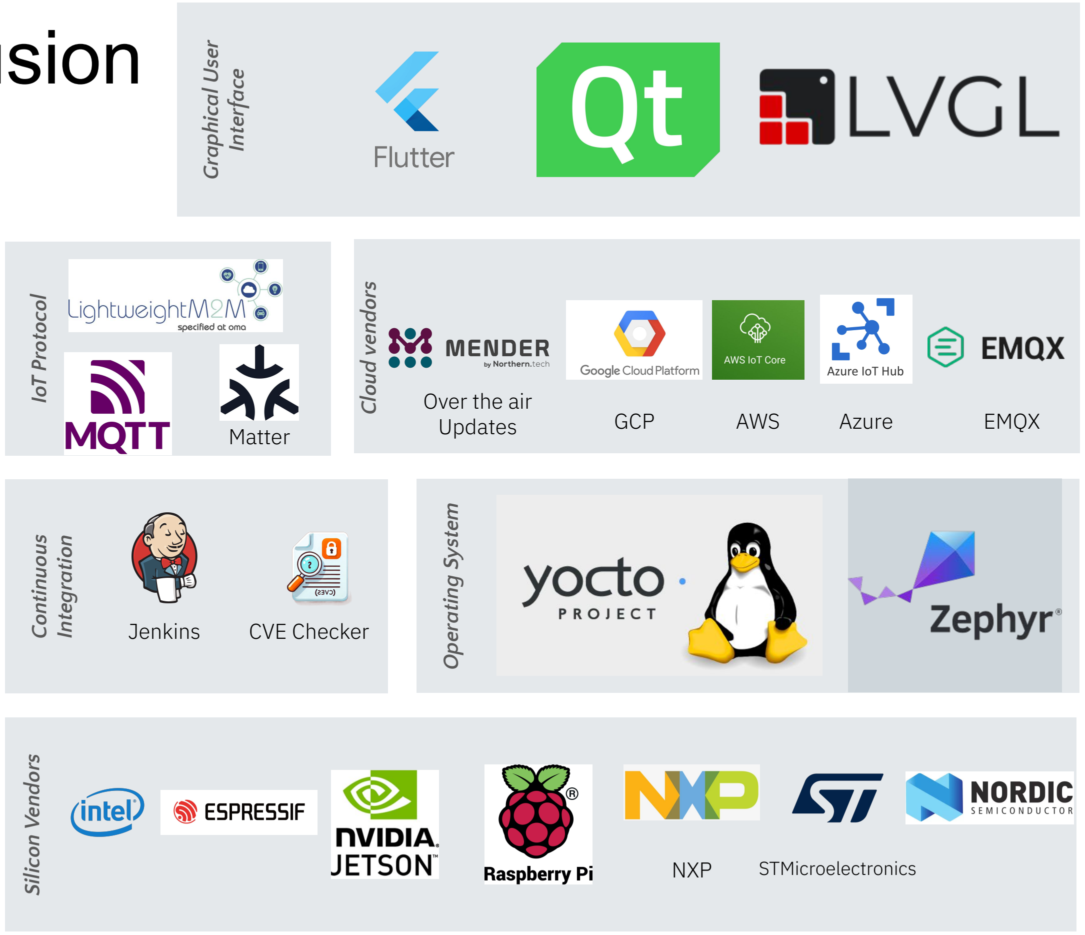
Depending on how much volume leverage and how big the contract with the commercial vendor, commercial vendor might not prioritize long term security, maintenance and your features roadmap nor change request. OEM might end up locked into roadmap & pace of vendor.

What Zephyr/Yocto Linux platform don’t provide out of the box, and additional effort required for commercialization.

- Functional Safety FuSa process for safety-critical systems.
- Verification of your real-time (soft or hard) requirements.
- System security design and continuous security (That includes your middleware and application components).
- Factory software download and provisioning secrets at the factory.
- Integration effort for tailoring the configuration and integration of platform modules (communication protocol, cloud endpoint, FoTA, peripherals .. etc)

Summary and conclusion

- The FOSS ecosystem offering for creating your next product is quite strong.
- On the right, FOSS projects of interest that offers easy integration in Zephyr and Yocto.
- Selecting silicon Vendor that supports and maintains reference integration with Yocto or Zephyr will save you effort on board bring up and will get you started immediately.
- For Zephyr: this awesome page to look for boards and available peripheral support, shields and samples.
- For Yocto: check the following silicon vendors page.



Thanks for listening

Open Questions