Lessons Learned: Embedded Linux Streaming with Open Source

Yocto · GStreamer · PulseAudio · BlueZ

Tokunbo Quaye – Principal Software Engineer
Intelligent Product Solutions
www.linkedin.com/in/tokunbo-quaye

System Overview

Custom Hardware

Not a dev kit — production SoC with tight constraints

Two Processor Generations support

iMX6 & iM8 systems

Strict Fidelity

Audio quality and is a product requirement, not nice-to-have

Long-Lived Devices

Field-deployed, non-technical users, must stay maintainable

Constraints Drive Design

Hardware & Integration

- Two processor generations; one codebase
- HDMI + line-out audio paths
- Companion tablet app over BLE

Product Requirements

- Playback
- Live & On-Demand
- Streaming + on-device storage
- Remote support is a product requirement

Lesson: Hardware diversity brings another layer of complexity to the software.

OpenSource toolset for building custom Linux OS from scratch for embedded systems

Yocto Advantage

Provisions for builds with diverse hardware targets.

Custom OS: Layer Strategy

Application Layer	Service recipes, app packages, runtime configs, init scripts
Hardware Customization Layer	Device-tree overlays, custom kernel config, downstream drivers
BSP Layer (Vendor)	Board support, low-level drivers, vendor-provided configs

Lesson: Derisk hardware interactions early

Yocto Lessons

Source Mirror

Maintain your own mirror — upstream tarballs change or disappear

BSP origin matters

Vendor BSP is not the same as Community BSP with the same version name.

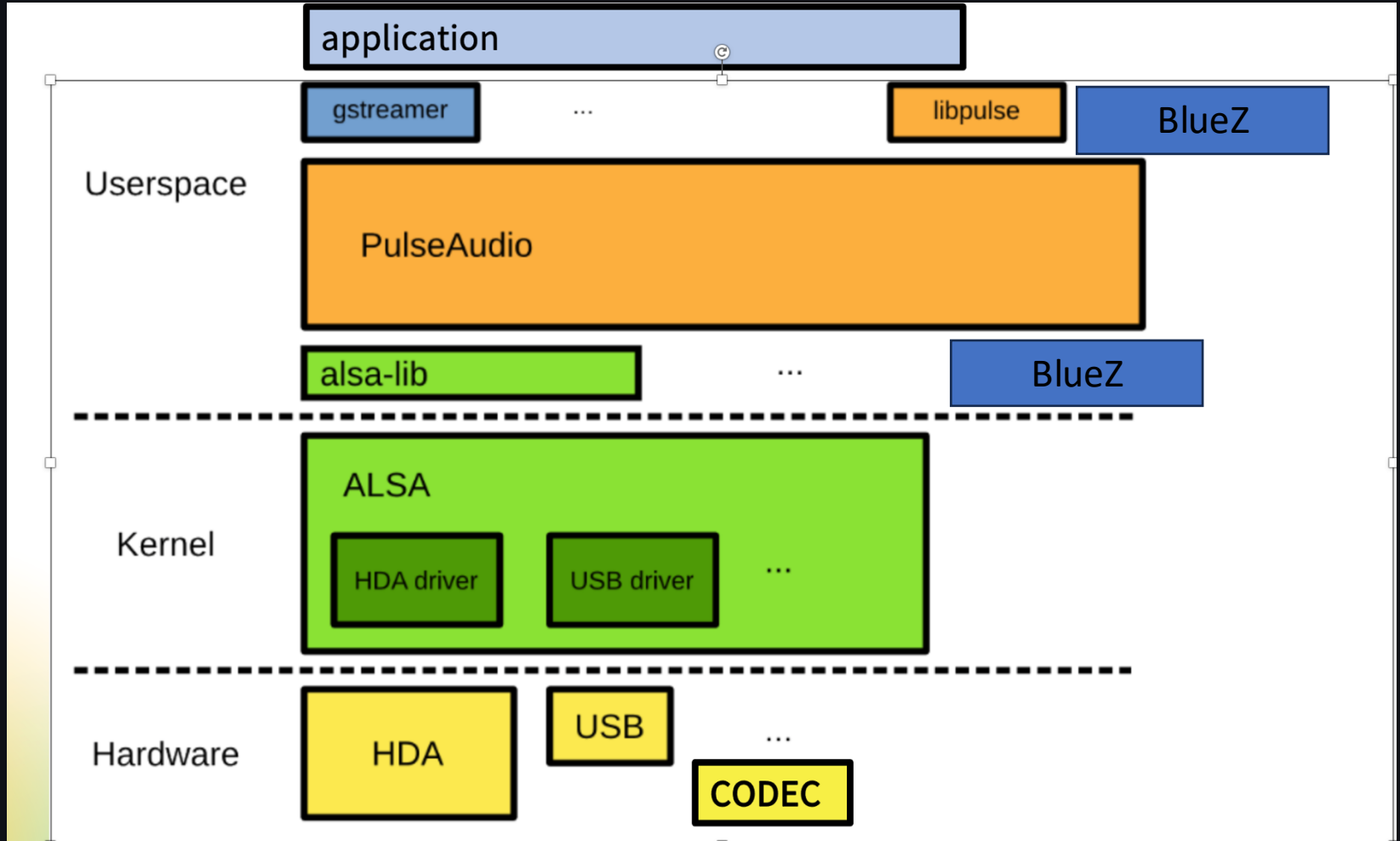
Major Version Jumps

Not trivial. Evaluate upgrades as they are released.

Out-Of-Order Bitbake Builds

Bitbake is multi-threaded and can result in out-of-order builds

System Audio Tech Stack



MultiMedia framework for creation , processing and streaming video & audio applications.

Pipeline Design Patterns

```
$ gst-launch-1.0 filesrc ! decodebin ! audioconvert ! audioresample ! pulsesink
```

- **Design around use cases, not codecs — pipelines serve product flows**
- Make caps explicit early — don't rely on autoplugging in production paths
- Separate branches with queue elements — decouple decode from downstream
- Latency is a deliberate decision — streaming vs. local playback differ

Gstreamer Lessons

Element Behavior Changes

Same element name, different behavior across versions. Never assume.

Software Decoders are an option

This is an option when hardware decoders are not present. Budget for CPU

Thorough Review of Release Notes

Between source and target version — even minor releases matter.

Debug pipeline on command line

Invaluable for isolating issues.

Performance Tradeoffs

Hardware decode vs. Software decode

HW Decode

- Lower CPU load — offloads decode to dedicated block
- May be unavailable mid-upgrade (driver churn)
- Not always portable across processor variants
- Requires vendor-specific GStreamer elements

SW Decode

- Works everywhere — portable across SoC generations
- CPU budget must be explicit — measure and cap
- **Define what degrades first: quality, features, concurrency**
- Set performance thresholds per use case — don't guess

PulseAudio is a sound server between the application and the sound driver

PulseAudio Lessons

- ! Pulseaudio resampling can silently degrade audio quality
- ! Bypassing this layer is sometimes the answer
- ! Combined sinks add complexity
- ! Setting kernel CONFIG flags can have audio impact

When to Bypass: PulseAudio → ALSA Direct

Consideration	PulseAudio	ALSA Direct
Routing features	Full (streams, sinks, policy)	None — app controls device
Layers / complexity	Higher	Lower — fewer moving parts
DMA/IRQ contention	Buffer policy abstracted	Must tune manually
Stability footprint	Larger attack surface	Smaller, more deterministic
Reversibility	Always available	Ensure config switch exists

🦻 *Make it configurable — the right answer may differ per deployment context.*

BlueZ is the official Linux protocol stack

Concrete Takeaways

- Budget time for big Yocto jumps
- Be willing to bypass layers — document the tradeoffs
- Separate BSP / hardware / app layers; test each in isolation
- Debounce BlueZ events; design for eventual consistency
- Keep a source mirror; pin all inputs to exact commits
- Design A/B updates + recovery from day one
- Read GStreamer release notes; CLI-test pipelines first
- Align PulseAudio rates; verify resampling in logs

BlueZ Lessons

Device Caching

Promotes faster BLE reconnects

Identity Surprises

randomization can cause unexpected identity changes

Bluetoothd connection is not atomic

Add debounce logic before acting on state changes — the OS may emit multiple events

Eventual Consistency

Bluetoothd - is eventually consistent — design your app layer to tolerate temporary inconsistency

Questions ?