Designing Your First Embedded Linux Device 2

Choosing Your Hardware

Nathan Collins | Senior Software Engineer, KDAB

KDAB

The most difficult decision you'll make when creating your first embedded Linux device has nothing to do with software; it's about the hardware. While it is possible to do a software upgrade mid-development without massive disruption and waste, attempts to make such changes in hardware development come at a much higher cost. So, you need to make some important decisions upfront about which CPU, board, and peripherals you want to use in your embedded Linux device.

This whitepaper looks at the numerous planning considerations that go into choosing your hardware to help you expediate your development process.

Should you use a custom or commercially available board?

The heart of your design – and the biggest cost and risk – is the main microprocessor system-on-chip (SoC). Until you have some experience with a particular SoC, you really want to leverage the experience of a vendor who builds them into pre-existing system-on-module (SoM). While that might cost a bit more per board, it lets your hardware and software teams get critical experience with a particular chipset before you start laying things out yourself. Once your company has hardware and software expertise behind a particular SoC, you could consider a bespoke board of your own.

Which silicon vendor is right for you?

When looking at chips you can use for your project, there are a huge number of vendors available. The biggest choice of hardware however is going to be your main central processing unit (CPU) and that's where we'll focus next.

Because so many chips are based on ARM designs these days, it makes it a bit easier to compare prices and features more directly. But while the CPU may be the heart of your project, what about

the surrounding board hardware? Many silicon vendors provide evaluation boards to help assess their chips like AMD, Intel, Nvidia, NXP, and Renesas. You might think it's best to go to the source since they know their hardware better than anyone else – and that is often true. Eval boards are often fully loaded, letting you explore the capabilities of the chip. That gives you a chance to explore your options before narrowing them down to the final hardware configuration.

You might even consider using a silicon vendor's board as part of your product. Whether this is a practical approach comes down to three main factors: cost, support, and longevity. Supplying eval boards as a product isn't a silicon vendor's mainline business. That means board costs are likely higher as they're only intended for very limited distributions needed for evaluation purposes. How well does the silicon vendor support the board – both with engineering advice as well as board support packages (BSP) – can also be a big factor. Even if you only anticipate very small production runs for your first product, the lifespan of vendor-supplied boards – or perhaps more importantly, how frequently they revise and make changes and how controlled that process is – can be a big factor in whether you can use them as the basis of your hardware platform.

From experience: Silicon vendor support can be critical

Most companies start their hardware choices by examining a huge matrix of hardware features. In those assessments, how well silicon vendors can support their customer is often overlooked. We worked with one customer who selected a chip that was a perfect fit for their application, and, for ease and convenience, they also used the vendor's Linux BSP. Unfortunately, the vendor was inconsistent in OS updates and generally poor in open-source management. That left the customer stuck with an older Linux 2.6 kernel even though they required some known patches. While the product was in demand throughout the pandemic, the lack of silicon vendor support to fix critical problems lead to a bad experience for the company – and their customers.

Ask your sales representative if you're considering using their board for production. They don't want disappointed customers or support issues, so they're almost always perfectly honest about whether they think their board products can be used for

production purposes. And as is often the case, if they prefer that you don't buy eval boards for production, they can steer you to the right distributor that can help you with a longer-term replacement.

What about picking a board vendor?

There are many companies that specialize in taking chips and creating fully capable SoM boards or embedded single-board computers around them. A few notable companies here are Advantech, Arrow, Avnet, Kontron, Toradex, TQ-Group, and Variscite – and there are many more. How do you pick among these many options?

This can be difficult because there are so many factors to look at for each board and each board vendor. Look at this list of options and see which of them applies to you.

- **Eval versus product.** In addition to evaluation boards, board vendors are more likely than silicon vendors to offer single-board computers that can form the basis for a company's product. Look in the board's specifications to see if "end product" is one of the expected uses. These boards often offer more engineering support and have longer product lifespans that are helpful if you incorporate them into your product. Board vendors may also offer design files for the schematics, layout, and bill of materials – as well as customization services – to allow you to transition more easily to creating your own customized board.

- **CPU support.** Vendors have many variants of a particular CPU line with differing maximum clock speeds, cores, co-processors, and graphics processing units (GPUs). This is one board component where it doesn't pay to go for the smallest option. Due to shortening schedules and feature creep, software engineers quickly eat up any spare processing power. Check the specific CPU offered on each board to make sure it's appropriately sized for your needs but if in doubt, go bigger.

- **Hardware features.** Probably the most obvious features to look for are the size and speed of on-board RAM and Flash storage. However, many other hardware components may be important to include on your board either during development or for final production, like support for HDMI (driving development or production screens), USB (debugging and connection to other hardware), SD cards (easily swapping in/out firmware builds), and other application-specialized buses or sensors.

- **Supplied OS.** Does the vendor provide their own customized version of Linux? Do they offer a Linux distribution like Ubuntu? Do they have a Linux deployment system like Yocto? Do they offer other OSes in addition to Linux? All of these questions let you evaluate whether the right Linux or Linux-like OS is already on the board or whether you need to dedicate engineering resources into getting it there and keeping your build updated. (We'll come back to whether you need to use Linux or a more specialized OS later.) Unless you've got Linux experts on staff, it's probably best to go with a vendor pre-loaded option when possible. It cuts down on your development time when your board vendor has the supporting software ready to go.

- **Software options.** Does your board vendor offer any special software options that can help either streamline development or help build and/or support the product? These might be things like containerized development or over-the-air updates allowing you to keep your in-field products bug-free. If you require software subsystems like these, getting them from the vendor can save you a ton of time building and maintaining them yourself.

- **Custom engineering.** Does the board vendor offer custom engineering services for items you can't yet handle in-house – like tweaks to a device driver or changes in the board startup? You might not know at the beginning of the project if you need custom engineering. Consider things that use the hardware in

an out-of-the-ordinary way, like fully using a chip's low-power modes, employing special techniques to achieve fast boot times, or relying on seldom-used driver features. These may be candidates for custom engineering. It's at least good to understand the scope of your vendor's capabilities for custom software support, even if you can eventually handle them yourself or outsource them to another software specialist.

- **Board longevity and support.** How long are a vendor's boards supported? Some vendors let you keep using a product in limited runs even once they've removed it from the price list. Others drop it after a set warning period. There's also the question of software support for deprecated boards. How long will obsolete boards be supported by their software team? Many of these questions are most relevant when you're putting software into long-lasting consumer goods that may require infrequent updates. If you're updating already deployed hardware that's already at a customer location – either via OTA, customer-driven refresh, or on-site technicians – how a vendor manages board support over the long-term is a concern.

- **Product roadmap.** Knowing how easy it is to move your product to a higher or lower capability platform can be important to some companies. Similarly, knowing whether a vendor stays on top of creating new hardware/software combinations is good to know. Although most companies don't publish their product roadmaps online, you may be able to get one through your sales representative, or at least make some educated guesses based on what is on the web with the current product lineup.

- **Cost.** The last consideration on our list is cost. While it's the easiest attribute to measure and one that's critical for your profitability, it's also the most misleading metric. The cost of a board does not incorporate your company's costs for any of the above items, most of which are very hard to quantify. Hardware has a piece price assigned to it, while any resulting

software development time from those decisions can only be estimated. And those estimates demand careful consideration of the requirements, potential architectural considerations, and skilled software practitioners.

Software development costs and timelines can often dominate product development; that's especially true in the embedded space where volumes are often smaller. So, picking a slightly more expensive chip that has big software benefits can offer huge benefits in the long run.

Picking the screen

Not all products have screens. Hosting your user interface on a built-in web server or controlling your device through a smart phone application is the best option for many devices. However, if your product needs a built-in screen, then you need to consider a few things to help you decide which screen makes the most sense.

Driving the screen: fancy or not?

The biggest question to answer is regarding the demands of your user interface (UI). If you plan on "fancy" graphics – a term that encompasses things like consistent animations, high frame rates, sophisticated rendering, 3D graphics, large screen sizes or "retina-quality" density – you are increasing the number of pixels that need to be regularly shifted out to the display. A modern fast CPU can handle these things but at the expense of bandwidth that could be used for other tasks.

If you do have "fancy" graphics, you should seriously consider getting a CPU that has a companion GPU that can offload the CPU from most of the rendering duties – even a simple bitblitter can help. In addition to the GPU, there is sometimes memory set aside for the display. This is faster than accessing external RAM, so it can be a limiting factor on how big or colorful of a display your product can support. Display technology isn't getting simpler,

and user interface demands are constantly increasing. If the GPU option on your CPU family isn't a big cost bump, it's worth it as a bit of insurance to ensure your product can handle more complex and visually stunning UI designs.

Companies often have in mind a touch UI that looks and acts like a smartphone with an interface that can dazzle their customers. Yet many embedded graphics systems available are comparable to that on a 10- or 15-year-old smartphone. Even with the hardware acceleration a GPU provides, these chips with older GPUs and limited memory bandwidth are far too underpowered to deliver to these expectations. If you have a high-resolution display, even a simple UI needs to move a lot of pixels to support it. We've worked with many companies who have struggled and failed to improve their sluggish UI – even deep software expertise, drastic UI simplification, and intense engineering efforts can only deliver incremental improvement if the hardware isn't up to the task.

To 3D or not 3D, that is the question

Most user interfaces don't really need 3D graphics since they're exclusively 2D or whatever 3D elements they use in the UI are just pre-rendered images. But devices such as manufacturing equipment, 3D printers, medical equipment, or CNC machines can really benefit by showing the user 3D visualizations. If you're going to use 3D in your embedded application, it's especially important to pick hardware with sufficient power.

A 3D scene may require a large number of triangles (250,000 is not unusual). 3D graphics also come hand-in-hand with larger screen sizes and refresh rates of 30 frames per second or more. These things make intense demands on the GPU, CPU, and memory pipeline, and many embedded processors are not truly up to the task. You may want to try several potential eval boards with a mock-up of your application to see what hardware works the best. Or get advice from consultants who have some prior experience in 3D rendering on different platforms.

Finding the right screen

Many physical characteristics of a screen are very application dependent. For example, things like refresh rate, temperature range, contrast, and visibility (ability to be seen in full sun or complete darkness) vary depending on if your product is an industrial controller used for round-the-clock factory shifts or a handheld portable measuring tool used in full sun.

Portable devices also need to worry about power consumption. Backlight isn't the only energy drain, and there are a lot of display technologies out there. Best to know the potential issues with the technology you're choosing. While OLED screens may look bright and not require a backlight, they're only minimally better than a standard TFT LCD screen for power, and they often have shorter lifespans that might not be appropriate. ePaper displays use minimal current but carry a host of limitations on color, resolution, refresh speed, and readability.

Finding the right GUI framework

You're nearly always better to go with an existing graphics framework rather than try to reinvent your own, and so many choices exist from capable and heavy to lightweight but limited. Frameworks that support multiple OSes and languages for typical embedded development are those like Qt, wxWidgets, Slint, Dear ImGui, or Crank Storyboard, while frameworks that target mobile and web such as React, Flutter/Dart, and Angular (and many more) can be options. If you're trying to make a cross-platform UI that works on your device as well as within a remote web-browser, you might also consider an HTML framework (of which there are many).

The choice of framework depends a lot on the type of graphics you want to display (as we mentioned in the "fancy" section) as well as the programming language environment.

Do you need to switch between applications? If so, you'll want a windowing system that innately handles this, so you don't have to

kludge something together. Do you need multiple screen types (a premium and an entry level for example) or different sizes as part of your product line? If the latter, you'll need two different layouts for the two different screen sizes – potentially with some controls hidden or accessed through additional dialogs.

Whatever type of screen technology you use, make sure it has reasonable bindings for your chosen programming language. By "reasonable," we mean that either the API for the framework is native (the toolkit should be in the same language you'll be using) or it is well-tested and complete for your language of choice. Having a framework that supports the key features you need is great – until you find out that the APIs for those features are missing in your chosen language.

Summary

Building your first Embedded Linux device is not easy. Hopefully this guide gives you a good feel for the many things you need to consider. Our engineers have deep expertise in all aspects of embedded product development so please don't be shy to reach out if you have any questions or need help at any point in the process.

This is the second whitepaper in a series of four that covers planning considerations and lessons learned in building embedded devices with Linux. Each whitepaper addresses a specific portion of the development lifecycle, so you can easily focus on the guide most relevant to your current stage of development. If you don't find the advice you need in this whitepaper, check out our first, third, and/or fourth whitepaper in the series.

View the four parts of this whitepaper online:
www.kdab.com/publications/embeddedlinux/

About the KDAB Group

The KDAB Group is the world's leading software consultancy for architecture, development and design of Qt, C++ and OpenGL applications across desktop, embedded and mobile platforms. KDAB is the biggest independent contributor to Qt and is the world's first ISO 9001 certified Qt consulting and development company. Our experts build run-times, mix native and web technologies, solve hardware stack performance issues and porting problems for hundreds of customers, many among the Fortune 500. KDAB's tools and extensive experience in creating, debugging, profiling and porting complex applications help developers worldwide to deliver successful projects. KDAB's trainers, all full-time developers, provide market leading, hands-on, training for Qt, OpenGL and modern C++ in multiple languages.

www.kdab.com