Cross-platform embedded system development

With CDP Studio you have a single development tool for all your customer specific embedded systems.

- ARM based embedded devices
- Re-use code across platforms
- Distributed systems
- High level development tools
- Project specific solutions
- Reduced development cost

The cost cutting impact of having a single development platform covering a wide range of hardware is significant. With embedded devices running Linux, tools, functional libraries, people skill-set, etc. is shared across systems. Now you can build embedded Linux devices much in the same way as CDP Studio is used for higher level control systems, removing a significant chunk of low level configuration.

With CDP Studio you can build a distributed control system with a combination of small embedded Linux devices, automation controllers, industrial computers, and even Windows desktop systems. All part of an integrated system, developed on the same platform.

The following will focus on the embedded end of this scenario to give some inspiration on how to use CDP Studio on small Linux powered devices and in embedded hardware environments, based on the current standard toolchains that ships with CDP Studio.

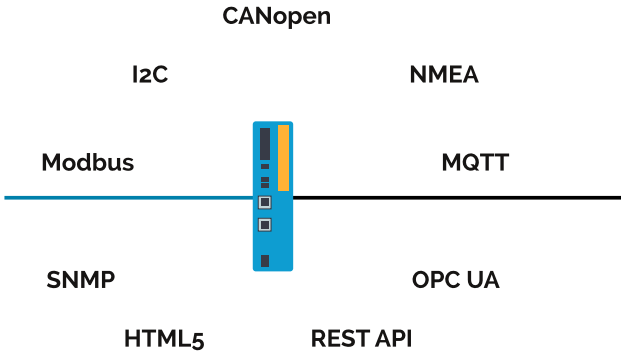
Embedded solutions

With the term "embedded Linux" we normally think of standalone appliances running a tuned, stripped down, Linux system. Embedded devices are designed to do a specific task, some also have real-time requirements. These devices are locked down with a given functionality, ranging from simple electronic toys, to marine navigational systems. The simpler products are delivered "as is", i.e. no ways to update the software or ways to interact beyond the operational user interface, while industrial systems tend to be both configurable and upgradeable.

For customer specific solutions, this is an opportunity to deliver tailor made solutions, even for low volume devices. This is where CDP Studio as a development platform provide the tools to design, configure and maintain systems, making such customisation a sensible business case.

Embedded device

For stand alone devices, there is just one run-time application doing its tasks, running on top of a trimmed down Linux OS. The end-user will not relate to CDP Studio as such, it is only used for development and maintenance of the software application. Examples here would be devices for data acquisition or dedicated controllers, probably configured by the end user via a web interface.



CDPStudio

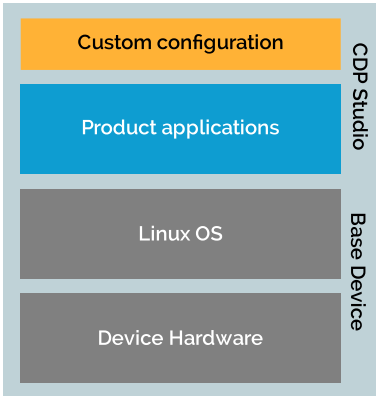
CDP Technologies AS
Hundsværgata 8
P.O. Box 144
NO-6001 Ålesund

www.cdpstudio.com
Tel: +47 990 80 900
info@cdptech.com

Customisation

Your product may be delivered in a standard version, but you also have the freedom to deliver custom variants. Modifying and enhancing products to fit specific customers or projects, will in many cases just be configuration changes, i.e. not require any changes in the underlying C++ code. Such changes may even be implemented by the project people, not involving R&D.

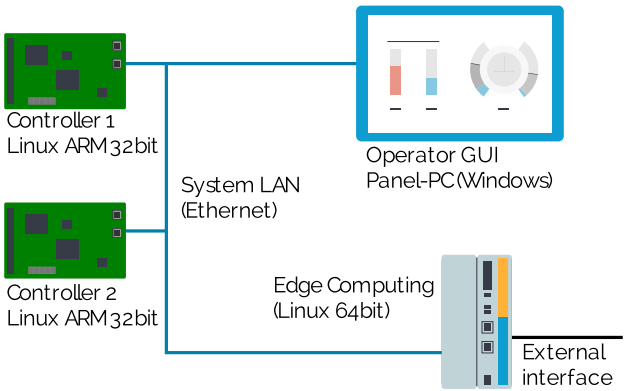
Add value to a product by offering customisation of the executable code



Taking this a step further, you could provide the CDP Studio development tool including your function library to system integrators or system manufacturers. Then your product is not just open for third party applications, but you provide a very accessible development environment for your customers to add their special knowledge or specific market segment functionality. Your product has added value!

Distributed systems

Even though embedded Linux products tend to be standalone devices, most will be attached to a network. CDP Studio has distributed system design built into the native application framework, i.e. a system is several applications working together. The applications may run on a single computer or distributed between controllers/computers on an Ethernet LAN segment. As CDP applications are abstracted from HW and OS, you may build a hybrid system solution of several low-level Linux controllers, a Windows Operator GUI, and a Linux IPC for the heavy signal processing. CDP Studio let you put functions where it makes most sense for system ruggedness, controller performance, combination with other software, storage capacity, etc..



System builders and integrators will then be able to focus on the solution, using a single development environment even if the hardware involved is coming from multiple vendors. The system as such has then increased hardware independence!

Hardware

When it comes to available hardware, CDP Studio currently has two ARM toolkits relevant for embedded devices:

- A generic ARM7 toolkit for Debian based systems.
- A dedicated ARM6 toolkit for Raspberry Pi based hardware running Raspbian.

There is also a range of compact SBCs (Single Board Computers) running Linux (or Windows) on x86 architecture which could be seen as embedded devices. Here we just use the generic Linux or Windows toolkits that comes with CDP Studio.

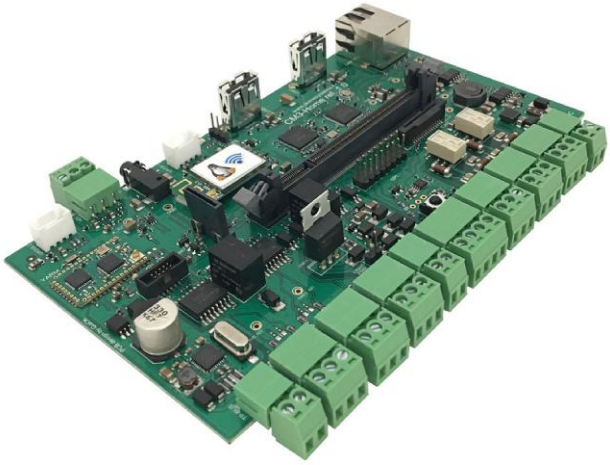
If we look at relevant ARM based hardware suitable for CDP Studio projects, the following examples gives a better picture of the possibilities. In general, if it runs a Debian derivative, in most cases your CDP applications will work, using the appropriate toolkit. Be aware that this is also linked to the CPU platform.

Raspberry Pi and derivatives

The Raspberry Pi ecosystem continue to grow and while the standard Raspberry Pi is not something you use for a mission critical control system, it is quite useful for prototyping and demos. CDP Studio has a dedicated toolkit for the Raspberry Pi, for use on the standard Raspbian Linux distribution.

There is also now an increasing range of small controllers using the Raspberry Pi compute module (Raspberry CM3), which is a Raspberry Pi 3 in a more flexible form factor, intended for industrial usage. Several development kits for the Raspberry CM3 module are available for device design.

The Revolution Pi is such an industrialised system complete with analogue and digital I/O etc. This boils down to a capable industrial Linux controller at a reasonable price. CDP Studio comes with an I/O server for Revolution Pi, so building a controller with bus attached I/O works straight out of the box. (https://revolution.kunbus.com/)



Another example geared more towards home control is the CM3-Home by Guiott (manufactured by Acme systems). The board fits in a DIN-rail enclosure and has a range of useful interfaces and proper 12-24VDC power, but no monitor connector, being a true headless embedded controller. (https://www.acmesystems.it/catalog_cm3home)

ARM Barebone hardware

If you are going deeper embedded, then there are multiple embedded ecosystems and development kits using the popular ARM architecture. CDP Studio will need a system that comes with a Debian derivative, either generic or HW specific builds using e.g. Yocto.

The BeagleBoard.org® ecosystem, the first completely open-source hardware community, now over 10 years old. They now have a range of boards coming from multiple manufacturers, which ship with Debian Linux . As the BeagleBone® is open-source hardware, integrating this into your electronics is easily doable. Variants of the BeagleBone@ may even be used directly in projects. (https://beagleboard.org/)

The normal approach would be to use a SoM (System on Module) including development kits or evaluation boards. An example here is the RoadRunner SoM from Acme Systems. With its single core Cortex A4 processor at 500MHz, the module is not very powerful, but still comes with a trimmed down Debian Linux. A neat package for small devices. (https://www.acmesystems.it/roadrunner)

There are several similar systems on the market, and with CDP Studio as a single development tool you will cover the full range from a tiny Cortex device to a powerful server workhorse. There is also the more powerful x86 based devices for specific markets, like the Marine computer below.

Courtesy of Recab AB

Toolchains, framework, and IDE

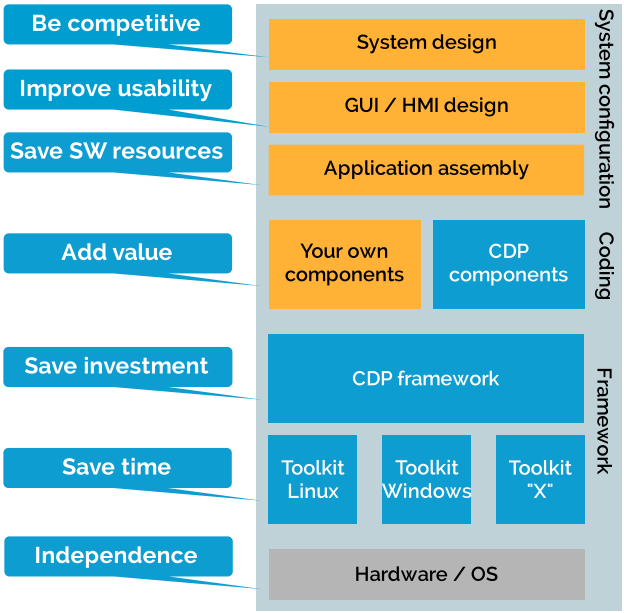
Using CDP Studio, the target applications run compiled code using linked libraries, which means the processing speed is as fast as it gets, hence utilising the hardware in an optimum way.

As hardware is increasingly more capable at reduced cost, size, and power consumption, application execution speed is not the only parameter. This is especially true for customer or project specific solutions; the development time of a system is the only place to save significant project cost.

CDP Studio is a complete development environment, implying that the maintenance of the overall tool is managed for you. CDP Studio is not just the IDE, but comes with a development framework geared towards control systems and real-time computing, a set of standard functions and protocols, and finally toolkits for several hardware platforms. If you need a product specific toolkit, we can build it as a separate add-on to the standard CDP Studio.

Cross-platform embedded system development

CDP Studio may be seen as a layered "stack" of functionality as illustrated in the figure.



Framework

One important aspect is the hardware abstraction achieved through a comprehensive framework, combined with complete toolchains for several HW/OS platforms. This is how you can develop on Windows, and even test most aspects of applications on Windows before they are compiled and deployed to the target hardware. CDP Studio covers the full range from Linux devices to Windows desktop systems, where functions are shared across all platforms. The hardware independence and the amount of time saved by the readymade complete toolkits included in CDP Studio, are great cost savers.

Coding

"Components" are the main building blocks in CDP based applications. A Component is basically a run-time C++ program, a component could cover anything from standard protocol implementations to customer specific algorithms. This is where CDP Studio removes much of the effort of developing in C++, as almost everything in the code except the logic itself is generated for you, using the graphical IDE. CDP Studio comes with an extensive library of standard functions, to which developers add their own components. This is where you build reusable added value.

System configuration

When it comes to configuration and assembly of the actual application, there is no coding involved. For customer projects, you need people closer to the end-customer, a different skill-set than C++ component development. In addition, CDP Studio has a full-blown GUI editor with a library of readymade widgets, to build graphical user interfaces, as a part of the solution.

Important is also the tools for testing and run-time analysis, all done from inside the IDE. As an example; you can tweak parameters on a running system while monitoring live signals and variables.

The final compiling and deployment on the target system is also controlled from the graphical IDE, which takes away a significant source of trouble and wasted time.

Configurable embedded systems, a business opportunity

With CDP Studio you can transform your application knowledge into function components as building blocks in your system design. Your knowledge can be distilled into your own library of CDP Components and reused as configurable functions without modifying the component C++ code.

The framework of a development tool like CDP Studio is important, as the developers should be "helped" into a common design structure, to safeguard quality and make the solutions better maintainable. Team collaboration is also a part of this to enable sharing and re-use of components, as well as speeding up large projects. A comprehensive tool, like CDP Studio, where the complete tool is released and updated as one package, is closer to an industrial mindset than handling a suite of independent tools. Industrial computing still differs to IT systems by longer update cycles and expected lifetime.

The freedom and power of software-based control systems is usable in industrial applications only if the development process is structured and the solution as such is possible to maintain for the lifetime of the system. With an efficient development system like CDP Studio, even single project usage of embedded Linux devices is feasible with a reasonable development cost.