Combination of Yocto and Qt creates future-proof embedded systems

By Christoph Kutzera, Garz & Fricke

The programming of embedded systems on an industrial scale makes particular demands of development. Ultimately, software products should be highly individual and performant while at the same time flexible and comprehensible. The GNU/Linux-based framework Yocto offers a solution here in combination with Qt.

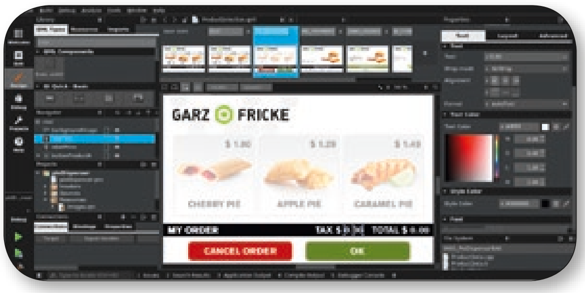

Figure 1. System and application developers work on separate levels. With the aid of the build system by Yocto, the parts are put together for the finished image of the target system.
(Source: Yocto Project / Garz &Fricke)

■ Limited computing power, special interfaces and high demands on reliability make the programming of embedded systems demanding. Essentially, developers have the choice between various Linux-based systems – excluding of course other more seldom-used base systems. However, off-the-peg systems only prove to be suitable in rare cases or increase production costs considerably. Other solutions have to be sought, such as Yocto. This enables tailor-made software to be created that not only does justice to the high demands placed on embedded systems, but can also be reproduced exactly even years later and thus securely maintained and developed further over the entire lifecycle.

The Yocto Project is collaboratively maintained under the umbrella of the Linux Foundation and is actively used and supported by prominent major players in the industry such as Intel, NXP and Texas Instruments. Here Yocto needs to be seen as a toolbox for individual GNU/Linux-based operating systems. In addition to the tools for creating the system, Yocto contains sample recipes. These are templates for the software creation and configuration. The special feature is that the recipes are organized as a layer system. Different configurations that are each relevant for hardware, specific applications or even only for specific components can be readily separated from one another. As a result, Yocto can be used to create individual, adapted Linux distributions – hence specific combinations of kernel, libraries and applications. This means that developers are not restricted to individual systems but instead are able to manage entire platforms in addition to the necessary software landscape. Linux inherently supports a great variety of different processor architectures, chip sets and mainboards and can thus be employed on numerous target platforms. One of the great benefits is that the platform developer can configure use of the available resources such as computing time and working memory very finely and precisely with the aid of Yocto and numerous compiler switches to obtain a performant overall system.

One example of the high level of optimization that is possible with Yocto can be seen in a medical product for which the Hamburg-based microelectronics specialist Garz & Fricke supplied a complete human-machine interface. In addition to the touch display, an HMI also requires an appropriate single-board computer (SBC). Initially the standard product Santino LT was involved. However, this was adapted with particular regard to the energy demand. As the finished medical product should be able to operate completely independently from the electricity mains for periods of hours up to days, the Deep Sleep and Standby modes had to function perfectly – yet neither of these is trivial. Implementation requires a detailed knowledge of the hardware in order, for example, to switch loads off and on again as well as set the correct voltages. Not only the customer from medical technology was able to benefit from the adaptations. Garz & Fricke also adopted the optimizations in matters of standby and energy consumption into its Yocto framework, where they will continue to be maintained for future projects as well. Work with Yocto offers yet another benefit: a high level of traceability in the case of adaptations and modifications. The comprehensive release notes with the detailed list of software and licenses used are particularly helpful in gaining medical approval, for example, also because the auditors ascertain at a glance which software or software components are contained in which version of the finished system.

Companies avoid whenever possible the dreaded situation of being locked in, in other words being committed to only one manufacturer, which can only be resolved at very great expense. However, there is little fear of this with Yocto. After all other hardware manufacturers also supply adapted Yocto versions as well, which contain the necessary individualized configurations and – in some cases proprietary – drivers. Another factor in favour of Yocto is that, if need be, the target platform can be exchanged at reasonable expense so that the costs of changing do not break the budget. The developer can remain in the customary software environment. Starting up with Yocto does require some work though. Fundamental configurations and the necessary tools such as compiler and linker are part of the Yocto package and are loaded subsequently during the first usage. In particular the comprehensive documentation contained as well as templates for beginners facilitate the

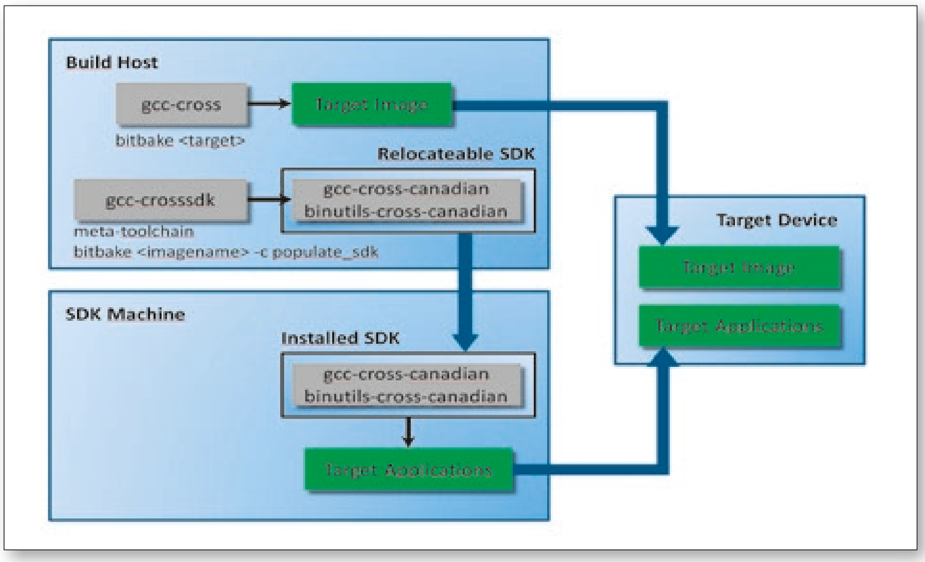

Figure 2. Qt Creator in design mode with external graphics inserted. The tool generates the necessary code automatically

(Source: Garz & Fricke)

start immensely. Manufacturers such as Garz & Fricke additionally give the customer best practice examples relevant to operation and step-by-step instructions. This approach and use of Yocto by system and application developers differs (figure 1). In both cases, they can use a wide range of languages and frameworks. Yocto offers the appropriate basis for translating the software sources from the bottom up. System developers will load the pure Yocto from the website of the software project together with additional components as necessary to match the hardware deployed, or they can also access an already adapted Yocto environment from the chip manufacturer. In contrast, application developers make use of prepared or adapted versions from the device manufacturers which have been created by their own system developers.

Configurations in Yocto are stored in what are called recipes. These recipes can be backed up and recovered and thus guarantee reproducibility even after many years. The recipes contain configurations that control how the software components are constructed. Garz & Fricke, for instance, therefore supplies a so-called board support package with every board that makes all recipes available to the developer in a completely open form. Embedded systems specialists thus enable their customers to integrate further software libraries and their own applications directly into the build process, or reconfigure and compile the kernel with only a few adaptations. This is where a major benefit of Yocto over conventional GNU/Linux distributions pays off. The changes made by system developers remain traceable. They can freeze the software project at a defined status. Even years later a project such as this can be resumed and developed further, such as in cases when security vulnerabilities must be eradicated. With a conventional distribution in these cases it is quite feasible that libraries and tools will have already evolved to such an extent that it is impossible to edit what is seen as ancient software from this advanced perspective, let alone translate it. In the field of embedded systems, which like vending machines or cash register systems have significantly longer lifecycles than a desktop system, this is an absolute no-go. Yocto preserves, as it were, all recipes as well as the development path to the finished binary image in such a way that when the preserved item is unpacked again, the desired binary can be recreated with only one command.

The Qt framework has turned out to be the suited partner in the design and programming of the graphic user interface for Garz & Fricke as well as customers and partners. At the same time, it is very fitting that Qt is among those involved in the Yocto project. Work with Qt based on Yocto usually starts with the Qt Creator, a development environment primarily for the programming of graphic user interfaces (GUI) with C++. Since Qt5, it has been possible to program based on QML and in some cases even HTML. Qt Creator opens three ways of programming for developers. One the one hand, they can of course write source code directly. Alternatively, they can use a so-called WYSIWYG editor, which in the case of QT is UI Designer. Here developers order graphic and other elements directly and see what they obtain: a user interface nested in several layers with underlying program logic (figure 2). In a third option for creating the GUI, UI Designer enables templates from other applications such as the image processing program Photoshop to be imported and developed further into the GUI. Many developers combine these three options. GUI developers do not necessarily meet with the actual operating system. This declarative programming in QML enables the interfaces to be described intuitively. Scaling is supported in this process, which makes developing more flexible because it is no longer necessary to define every element exactly to the nearest pixel from the outset. Furthermore, graphics and video are very well supported, which greatly simplifies embedding into the graphic user interface. Added to this is OpenGL support, so that, with a suitable board and graphics chip, the display of the GUI, effects and animations as well as videos benefit from hardware acceleration. During development, Qt Creator helps developers with extensive opportunities for debugging and optimization. In this way, it is possible to integrate tests into the development. Furthermore, the development environment allows the program run to be stopped at any time or at defined points. To achieve the best possible display performance, Creator provides analytical tools to help developers identify heavyweight elements or unnecessary iterations.

Another example from practical applications shows how Qt simplifies the work of the developer. A machine that Garz & Fricke were to equip with displays including control system was originally planned with two screens. Despite the existence of a functioning prototype, a decision was made to dispense with one of the displays. Instead of having to restart the software and GUI development, the developers were able to implement the change in concept to just one screen inside a very short time within the existing software project. During this process, the changes to specific output graphics accounted for the greatest share. The necessary adjustments to layout, on-screen instructions and control logic were able to take place swiftly thanks to the good programming model in Qt and QML.

Qt uses dual licensing, which is something that companies should note. Basically, the entire framework is open source and is subject to the LGPLv3. Accompanying this is the fact that the person using Qt is subject to certain obligations. One of these obligations is that Qt cannot be compiled statically into the system binary but instead can only be dynamically linked. If it is to be integrated statically, the entire source code of the binary also must be disclosed. Furthermore, it must be possible for the buyer to carry out changes in the software components subject to (L)GPL v3. However, this is precisely what users should not be able to do under certain circumstances so as, for example, to guarantee the function of the product or maintain company secrets. In such a case, the company involved in development can make use of a commercial Qt licence, which then releases it from this and other obligations. ■