Overview: Processor Expert

Posted on **October 18, 2015** by **Erich Styger**

10 Votes

In “Overview: From Snippets to Code Generation” I discussed several tools used in my development process. On tool which helps me a lot to get things done is Processor Expert. In this post I’ll give an overview about this tool and reasoning for the pros and cons of using it.

— Processor Expert

Overview

In this article I go over the concepts of Processor Expert and how I’m using them in my applications. I have used it in many customer and research projects over the last years. As a tool might not fit every use case, I try to list the pros and cons how I see them.

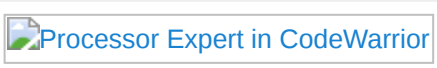
History

I don’t know the full history of Processor Expert by any means, so here is just what I know and was able to gather: Processor Expert is a tool who has been developed by Unis, starting around 1994 in Brno, Czech Republic. I started using that tool around 1998 and later with the ‘classic’ CodeWarrior using the Motorola/Freescale HCS12 and HCS08 microcontroller. At that time, Processor Expert technology was independent and available for non-Motorola/Freescale processors too, e.g. for Fujitsu. In 2001, it has been awarded the European IST price.

💡 *The European IST price is mentioned in web documents, but this IST organization/web page seems not to exit any more?*

The technology has been acquired in 2008 by Freescale and supports Freescale only processors by now: HCS08, HCS12(X), DSC, ColdFire, i.MX and Kinetis.

At the beginning, the tool was available as a ‘standalone’ version (without IDE), but then has been integrated by Metrowerks into CodeWarrior IDE as plugin.



— Processor Expert in CodeWarrior

Today it is available as plugin in Eclipse (e.g. Kepler, Luna) or integrated in Freescale Kinetis Design Studio, and runs on Windows, Mac OS X and Linux.

— Processor Expert in Eclipse

Pros	*Cons*
Develop a working application very fast.	Needs time to learn the tool.
Has built-in knowledge base, sometimes better than the reference manual.	A solid understanding of microcontroller is still required.
Graphical User Interface.	Needs ressources on the host to run the GUI
Integrated into Eclipse and CodeWarrior IDEs.	Other non-Eclipse IDE's like Keil or IAR need to use it as external tool which is not that user friendly.
Broad range of Freescale devices supported.	For Freescale only

Concept and Flow

The concept of Processor Expert is that it I can select 'components' from a **Component Library** to use and combine them in my application. Components are like 'Lego Blocks' with interfaces which can be put together and which can be configured. Based on this configuration, it then uses a **Code Generator** to produce normal C/C++/Asm source files.

— Processor Expert Flow

Beside of the source files, that code generator produces report and documentation files. Together with my source files they get compiled/linked to build the ELF/binary file for my board.

Pros	*Cons*
Ability to do rapid prototyping with components from the library.	Depend on availability of component for devices, not all Freescale devices are

	supported.
Source files optimized and generated based on my settings, the generated code is static.	Safety coding rules can prevent any 'generated' code. Using a new version of the tool can affect code generated, needs re-certification of application/code.
Sharing of common code and library functions.	Code generation especially for many modules can slow down a full build of the application.

Component Library

The component library has 'blocks' for reading in analog values (ADC) or toggling pins (Bit I/O) up to more complex things like communication stacks (USB) which itself can contain several sub-components. Components can be very hardware specific (like a UART), or can be pure software components like one to implement a ring buffer.

— Components Library in Eclipse

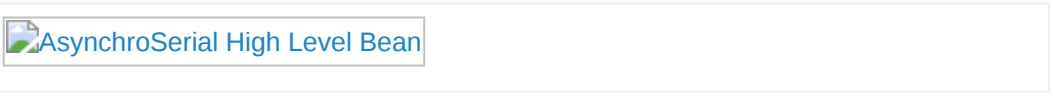
An installation of Processor Expert comes with many components included. It is possible to create your own components using the CDE (Component Development Environment) framework.

Pros	*Cons*
Reusing common blocks as a library in application.	Up until recently (Processor Expert for Kinetis v3.0.0) it was not possible to use multiple repositories.
Browse and search for functionality/components.	If functionality is not provided by Freescale, need to add/create own or 3rd party component.
Broad range of components, typically cover on-chip functionality.	Very few 'off-chip' components available like sensors or other external components. Freescale only provides components for Freescale devices.
Graphical user interface with online help with 'help on component'.	Lack of documentation, tutorials and example projects.

High Level Beans, Logical Device Drivers and SDK Components

In Processor Expert there three different types of components:

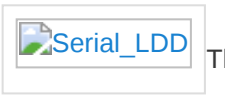
1. **High Level Components** or Beans (HLB): These were introduced first. They use a simple API. E.g. to send a character on the UART I call AS1_SendChar('a'):



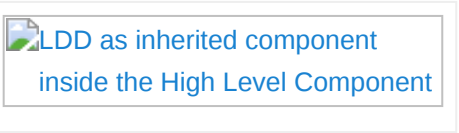
— AsynchroSerial High Level Bean

This can be easily used for bare metal and application with an RTOS. Of course if using a device from multiple tasks there has to be a synchronization in place. The biggest advantage of the 'Beans' is that they have a common API across different architectures, so I can use the same API/components across say HCS08, HCSC12, DSC and ColdFire devices. So porting an application from one architecture to another typically is a matter of hours.

2. Later on the **Logical Device Drivers** have been introduced with the introduction of Freescale Kinetis Devices. These components have the "_LDD" at the end of the name. The difference is that now a 'device handle' needs to be passed in each API call:

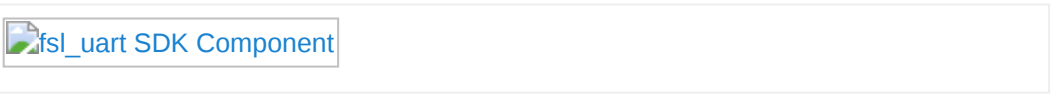


These components are not that easy to use and not as efficient as the High Level Beans as there is always that extra parameter. While that 'device handle' is a good thing for some applications, it is an overkill for bare metal applications or where a device is used by a single RTOS task only (which in many cases is the default use case). The biggest issues with the LDD is that their API did break compatibility with previous (say S08 and S12) projects, as only available for Kinetis. That's why soon after the release of LDD components Freescale had to provide High Level Beans for Kinetis too to provide a compatibility API. So with using inheritance both API's were offered:



— LDD as inherited component inside the High Level Component

3. With the introduction of the Kinetis SDK a new type of components were introduced: the **SDK components** which have the "fsl_" in front of their name.



— fsl_uart SDK Component

The SDK components itself do not generate the full driver. Instead, they generate the configuration structures for the Kinetis SDK. The SDK is using a HAL (Hardware Abstraction

Layer), and for this there are as well '_hal' components offered and which are used inside the 'fsl_' components. Even if these components do not generate the code, they offer an easy visual way to use the Kinetis SDK.

Pros	*Cons*
Simple API for High Level Beans. Easy to use and learn.	Application code needs to deal with synchronization
Extended set of functionality e.g. for Timers or the Flex Timers on Kinetis.	Overhead and complexity of LDD which is not needed for many applications. Hard to use/learn
Generic Device handle for LDD, easier to use it in an RTOS environment which uses device handles (e.g. MQX).	LDD API not compatible with previous applications using HLB components.
'Static' configuration and code generation with SDK components. Components are for 'configuration', not code generation. Easy and visual way to use the Kinetis SDK and HAL API.	SDK components break again with previous projects. Components/projects cannot use/mix different components, no portability API. SDK itself adds complexity and overhead.

Methods, Events and Properties

I can use components from the Component Library. A component can use **Methods**, **Events, Properties** and **Inheritance**:

- **Methods**: Functions/procedures of the driver. E.g. SendChar() for an UART component
- **Events**: Hooks or interrupt events. E.g. OnRxChar() interrupt hook for an UART component
- **Properties**: Settings of the component which affect the behaviour of the driver.
- **Inheritance**: Multiple components can be combined or using links/interfaces to other components, in an object-oriented way. For example a UART component can inherit from a low-level UART component and build up a new interface with new functionalities.

— AsynchroSerial Component

Methods and events can be enabled or disabled it not used (marked with a small 'x') in above screenshot. Methods which are disabled are not generated, and the component code generator might optimize the driver based on what is enabled/disabled. For example if there is no 'GetStatus()' method needed, then the component might not need to store a status somewhere at all.

For the components, there is a view to change and inspect the properties: for the UART component for example I can turn on interrupts and assigns the pins to be used for Rx and Tx:

— Asynchroserial UART Component Properties

💡 *I'm showing the 'classic' view in above screenshot. There is a 'tab' view (default) too, but that tab view in my opinion is much harder to use, so I recommend to use the 'classic' view. See "Switching between 'tabs' and 'no-tabs' UI in Processor Expert" for details.*

Basically, this gives me a graphical front end for my UART driver. Based on my settings, it will configure and produce software Files for me:

— Generated UART Driver code

The source code for sending a character over UART will look like this:

```
/*
** ===================================================================
** Method : AS1_SendChar (component AsynchroSerial)
** Description :
** Sends one character to the channel. If the component is
** temporarily disabled (Disable method) SendChar method only
** stores data into an output buffer. In case of a zero output
** buffer size, only one character can be stored. Enabling the
** component (Enable method) starts the transmission of the
** stored data. This method is available only if the
** transmitter property is enabled.
** Parameters :
** NAME - DESCRIPTION
** Chr - Character to send
** Returns :
** --- - Error code, possible codes:
** ERR_OK - OK
** ERR_SPEED - This device does not work in
** the active speed mode
** ERR_TXFULL - Transmitter is full
** ===================================================================
*/
byte AS1_SendChar(AS1_TComData Chr)
{
```

```
  if (SCI1S1_TDRE == 0U) { /* Is the transmitter full? */
    return ERR_TXFULL; /* If yes then error */
  }
  SCI1D = (byte)Chr; /* Store char to the transmitter register */
  return ERR_OK; /* OK */
}
```

So this is not much different from any other UART driver. The difference is the knowledge and the consistency checking Processor Expert is doing. For example it knows which pins are available for the Rx function, and which pins are already used by something else:

— UART Pin Assignment

There is as well the CPU view which keeps tracks about which blocks and pins are used:

— Processor Expert CPU View

The other thing is that it proposes me possible UART Baud values I can use:

— Baud Selection present in UART

For this it keeps track of the internal clocks and clock paths:

— Clock Path to UART

All this information would be present in the device reference manual too. But it is much easier and faster to use the AsynchroSerial/UART component:

1. Add component to project

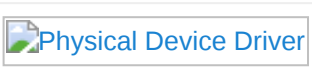
2. Select UART device (e.g. UART0)
3. Choose Rx and Tx pins
4. Specifiy initial UART Baud rate
5. Generate the driver/configuration
6. Use it in my application

If I'm not happy with the generated code, I have the option to 'freeze' the code generation of a component. See "Disable my Code Generation".

Pros	*Cons*
Easy and simple way to have a driver.	Still need to understand how the hardware works to some extend.
Graphical UI for configuration.	Requires IDE integration (Eclipse).
Consistency checking prevents wrong or bad settings.	Consistency checking requires time, slows down the tool. Sometimes 'rekursive' dependencies are not properly resolved.
Generates driver code and example code 'which works'.	API and coding style might not match own preferences. Because the driver need to deal with many different use cases, it is not the most efficient one (code size, speed).

Physical Device Drivers

In case the generated functions and methods are not enough, there is a lower layer of API which can be used: the Physical Device Drivers (PDD, see "Low-Level Coding with PDD (Physical Device Driver)"). Basically the PDD are low-level macros which can be used to access the low-level functionality of the device. Because the PDD macros are tight to the device used, they are not much portable between different devices.



— Physical Device Driver

Pros	*Cons*
Small and tight code.	Hard to understand API and how to use it.
Access to low-level peripheral function.	Macros are device dependent, not portable.
Graphical UI to drag&drop macros/functions	Lack of documentation.

Inheritance

Another great functionality of Processor Expert components is their ‘inheritance’ model: A component can inherit functionality from other components. The following shows the LED component which is inheriting functionality from a ‘BitIO’ component:

— Component Inheritance

The LED function On() (to turn the LED on) will use either the BitIO ClrVal() or SetVal() to change the logic level of the pin, according to the LED settings how the cathode/anode of the LED are connected to the port pin. This inheritance scheme is very powerful and enables component sharing. Processor Expert has as well the ability to ‘link’ to other components. For example the LED component can link to a Shell component, and that way automatically enable command line support for that component:

— Linked Components

Pros	*Cons*
Ability to turn on/off functions to optimize the code.	Sometimes unclear what the impact of enabling/disabling functionality will be.
Easy way to generate code “which works” or can be used as reference implementation.	Code sometimes not ideal, or not very well documented.
Automatically checks for dependencies, configures the pins and clocks for me. Checks for conflicts.	Still need to understand the microcontroller. Making bad settings will result in bad results.
I can disable code generation and do my own changes.	Freescale provided components are ‘closed’: I cannot edit or change them.

Component Development Environment

The component framework in Processor Expert allows me to create my own components. that capability is free of charge (no license required) if I’m not going to sell the components created. This is the ‘Community Edition License’, and if no other license is found, this text in the Eclipse Console View is printed:

```
CDE: no license found, using Community Edition License
```

I wrote a tutorial "Tutorial: Creating a Processor Expert Component for an Accelerometer" how to create my own components. Newer Processor Expert releases come with a Java based/Eclipse plugin to create/change your components. In classic CodeWarrior there was the 'BeanWizard' (standalone), but this one required a paid/Professional CodeWarrior license.

Using CDE, I can create components like the one on McuOnEclipse SourceForge site, or like the components provided by Freescale. With one exception: it is **not** possible to create Processor components:

— Processor Components

Pros	***Cons***
Can create my own components.	Not easy to learn, takes a lot of experience.
Free license (Eclipse plugin)	Cannot sell components, cannot create Processor components. Classic BeanWizard is not free of charge.
Graphical wizard and user interface to create components.	The CDE plugin in Eclipse is not very mature and not as easy to use as the 'classic' BeanWizard.

Portability

The other big advantage of using components is: as the component interface remains the same, and because Processor Expert is available on multiple Freescale platforms, it is very easy to move projects between different derivatives. For example I have my INTRO Robot application running on Freescale S08, ColdFire and Kinetis (ARM), all with the same set of components. So Processor Expert makes it really easy to move from one device to another (e.g. Kinetis KL25Z to Kinetis K64F) or from one architecture to another (e.g. HCS08 to Kinetis KL25Z), or from one toolchain to another (e.g. CodeWarrior to KDS or to any other Eclipse IDE).

For example this is a project for an S08 (Freescale S08 16bit microprocessor) on CodeWarrior which uses multiple components including an LCD:

— SRB Nokia Project (S08, CodeWarrior)

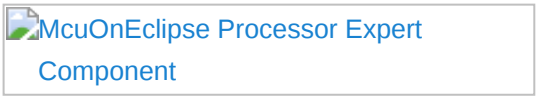
The same functionality/project on a 32bit Kinetis (ARM Cortex-M0+) on Eclipse Kinetis Design Studio:

— Nokia project on ARM (KL25Z, Kinetis Design Studio)

Porting such a project is a matter of an hour or less:

1. Create a new (empty) project for Processor Expert for the new target
2. Copy-Paste the components to the new project (see "Copy of Processor Expert Components")
3. Change the component settings to use the new pins on the new board.
4. Copy the application code
5. Generate Code, build and test🙂

A good example how Processor Expert components can be portable across different toolchains, architectures and IDE's is the FreeRTOS Processor Expert component maintained in the McuOnEclipse project:



— McuOnEclipse Processor Expert Component

A single component with a graphical user interface to configure FreeRTOS:

- 9 Compilers and 6 Architectures: Freescale chc12, chc08, DSC, ColdFire V1, ColdFire V2 and ARM compilers, GNU GCC ARM Embedded (launchpad), IAR, Keil
- 7 IDE's: CodeWarrior (classic) 6.x, CodeWarrior (eclipse) 10.x, Kinetis Design Studio, Driver Suite 10.x, Atollic TrueStudio, Emprog Thunderbench, NXP LPCXpresso
- 3 component types supported: HLB (High Level Beans), LDD (Logical Device Drivers) and Kinetis SDK.

One component to rule them all🙂🙂

Summary

I know this is a rather long post, and I hope it gives an overview for newbies what Processor Expert is: A component based framework which encapsulates software with a graphical configuration front end. There is much more behind this framework, including automated build and generation support. Of course no tool is perfect, and no tool fits every need. The problem I see is

that the framework should be faster especially for large projects. But the biggest disadvantage (maybe Freescale might say that this is the key feature😉 that it exists only for Freescale devices. So once someone has explored the power of it, it will be hard to move to another vendor.

To me, Processor Expert is like C++: it takes some learning and time to master it, but then it is incredible powerful. Sometimes I feel like at the old days when C/C++ compilers got introduced, and everyone was using assemblers: high level language was considered as slow, inefficient and "I can program this better!". But today there is no argument about using compilers because they have both improved and second we need to be more productive. And everything else looks like stone age😉.

Happy Experting🙂

LINKS

- Unis company: http://www.unis.cz/
- Processor Expert web site: http://www.freescale.com/ProcessorExpert
- Disabling Processor Expert code generation: Disable my Code Generation

RELATED

Tutorial: IAR + FreeRTOS + Freedom Board

Mother of Components: Processor Expert with NXP

The Making Of RTOS Processor Expert

In "Boards"

Kinetis SDK V2.0 Projects
In "ARM"

Components
In "CDE"

This entry was posted in **ARM**, **CDE**, **CodeWarrior**, **ColdFire**, **CPU's**, **DSC**, **Eclipse**, **Embedded**, **Embedded Components**, **FreeRTOS**, **Freescale**, **HCS08**, **KDS**, **Kinetis**, **Processor Expert**, **S12**, **S12X**, **SDK**, **Thoughts** and tagged **Eclipse**, **Embedded Component**, **FreeRTOS**, **Freescale**, **Processor Expert**, **software**, **technology**, **Thoughts**, **Tips&Tricks** by **Erich Styger**. Bookmark the **permalink [https://mcuoneclipse.com/2015/10/18/overview-processor-expert/]** .

About Erich Styger

Embedded is my passion....
View all posts by Erich Styger →

10 THOUGHTS ON “OVERVIEW: PROCESSOR EXPERT”

irwinsingh
on **October 18, 2015 at 19:40** said:

Hi Erich,

Very timely and great overview!

I have been using Processor Expert from Unis days and I really love it. Despite relatively long history using this tool, I have never felt being able to use PEx with ease. Some thoughts on that:

1) Whenever I would start getting comfortable with the tool, Freescale would make drastic changes to it. Akin to a large meteor hitting the earth and life had to restart. Who knows what is in store once NXP merger is complete😟
2) There has been reluctance in adoption within Freescale organization. On one hand it is good to see PEx examples in KDS, on the other hand, during 2015 FTF very little was talked about PEx in technical sessions – probably for the first time.

I agree with you that once you start using PEx, not using it is like going back to the stone age. I sincerely hope that it remains a focused enablement tool under NXP umbrella too with more adoption and promotion by new organization.

Thanks,
-Irwin

★ Liked by 1 person

Erich Styger
on **October 26, 2015 at 08:45** said:

Hi Irwin,
many thanks for your thoughts. I think it is a great technolgy, and I noticed your observation at FTF too when we talked there. Processor Expert to me was first in the market with that new approach, and when you look around at this time every other vendor has realized that and has now a similar technology in their portfolio. I whish there would be something like this available as a cross-vendor technolgoy/tool, like Eclipse so it would not be vendor centric.

★ Like

neilh20
on **October 21, 2015 at 19:13** said:

Great overview. I've come to know PE through your fantastic blog. I'ts really helped learn the Kinetis devices, particularly the clocking. I really appreciate some of the realistic examples you've used to show how the silicon can be tickled.
Unfortunately what stands out is lack of usage/recommendation by Freescale Support, and recently I've come to doubt whether Freescale with their ecosystem of partners are upgrading their ability to be able to educate people using their product. So fantastic tool, but requires some Freescale commitment that I don't follow. I've also used mbed – which was incredibly productive till I went off the edge of supported processors, core libraries. So its only good if the processor/library is supported – and that makes it a training/prototyping tool.
I'm also trying out the STM32cubeMX – generating drivers and a simpler visual of ARM chip pins – its easier to relate to and simple module code output. Still exploring
My method of using PE/KSDK1.x (and other tools) has been to focus on edge issues that are a challenge to me – currently that is USB HOST CDC/MSC . I'm looking for a connected device (IoT) that is easy for a customer to assemble components in the field through PlugNplay USB. The PE libs provide quick ways of prototyping solutions for FatFS/SPI

microSD, SPI Flash as do some of the KSDK_1.3 examples that can be imported into KDS3.
For Iot users (in low power nonLinux) the North Bound internet traffic connection is painful – wireless WiFi, BTLE, Cellular Xyz flavour, terrestrial country specific = and USB PlugNplay wireless devices could solve this.
So I try out code libs, compare against KDSK1.3 examples, keep a long list of ways that don't work and try and find out ones that do work. Its been very slow and unproductive.
The end goal is a stable code base running on processors configuration with enclosure that works for an end target customer base. Stable code to me is built slowly, doing integration tests – and for that PE is not useful.
PE helps identify modules that can be unit tested for functionality/hardware integration- but then have to be moved out into a source managed base. PE works to help define a board support package.
A really valuable environment is a developing knowledgeable community like the forum.microPython.org – with a distributed knowledgeable open source team, defined accesible development environment (make/github) that has a deep knowledge of the software stability and is able to add functionality and maintain stability as it goes along around their pyBoard.
Eric really appreciate the blog and sharing methods, and training the next generation in great ideas.

★ Like

Erich Styger
on **October 26, 2015 at 08:50** said:

Thanks for all your thoughts and very good points. Yes, good point on mbed too: it is really a 'walled-garden' and nearly impossible to exlore it outside the supported platforms which are basically demo boards. Not sure if ARM will be able to turn this around with their IoT strategy. And I agree wtih your observation on the drivers (e.g. USB), that was the reason why I wrapped up my own USB component around the Freescale USB stack to make it easy to use. While Processor Expert has good support for the low level peripherals, what is lacking is the support for external devices. The same problem I see with the Kinetis SDK: there is no support/driver to reach out to the external world of devices. What I wish is that there would be a good driver library for all the actuators and sensors out there, at least for the most common ones.
I have looked as well at microPython.org: that project looks very promising, indeed.

★ Like

Pingback: McuOnEclipse Components: 08-Nov-2015 Release | MCU on Eclipse

Pingback: Using FRDM K64F / KL25Z with arduino IDE | ultrano

Manuel Malagon
on **February 20, 2016 at 10:16** said:

Hi Erich
I'm confused with the SDK components. If this components do not generate code, what's the point? How is the SDK used then if there is no code generated? Or I think I didn't quite understand the concept. If possible, could you explain a bit further using for example the UART? Thanks!!

★ Like

Erich Styger
on **February 20, 2016 at 11:20** said:

Hi Manuel,
in case of the Kinetis SDK, the Processor Expert component is not creating the full driver code (as it would for non-SDK components). Instead, it creates only the files for the configuration of the SDK driver. You can see the files if you right click on the component and then use the Open File menu.
I hope that makes sense?
Erich

★ Like

Pingback: Mother of Components: Processor Expert with NXP Kinetis SDK V2.0 Projects | MCU on Eclipse

Pingback: Tutorial: Blinky with NXP Kinetis SDK V2.0 and Processor Expert | MCU on Eclipse