RENESAS
BIG IDEAS FOR EVERY SPACE

# BASICS OF THE RENESAS SYNERGY™ PLATFORM

Richard Oed

2020.04

# CHAPTER 2
# DETAILS OF THE RENESAS SYNERGY™ SOFTWARE

## CONTENTS

# 2 DETAILS OF THE RENESAS SYNERGY™ SOFTWARE

**What you will learn in this chapter:**

- What the different components of the Synergy Software Package (SSP) are and how they are layered.
- Details of the different layers and how they work together.
- Specifics of the X-Ware™ middleware and the ThreadX® RTOS.

The best microprocessor cannot unlock its full potential if either the software running on it is not up to the task, or the software ecosystem is too complicated to be useful to the developer. A well thought-out concept like the Synergy Platform helps to create applications which are easy to develop, simple to maintain, and which make the best out of the performance and the features of the microcontroller (MCU).

The simplicity found in the Synergy Platform was attained by making all major parts, MCUs, software, tools and kits and application solutions work perfectly together.

The Synergy Software Package (SSP) was specifically optimized for the Synergy MCU architecture, which, in turn, was developed with the software in mind. The SSP integrates application frameworks, functional libraries, the hardware abstraction layer (HAL) drivers, and, as the basis for all of that, the Board Support Package (BSP), together with the widely used ThreadX® Real-Time Operating System (RTOS) and the X-Ware™ middleware from Express Logic, which are pre-licensed for unlimited development and production usage if used together with the SSP. This results in a single and easy to use software package where all the functionality can be accessed by a simple and robust API (Application Programming Interface).

The complete SSP, even the parts created by Express Logic, are fully supported by Renesas. This means that you, as a developer, will have only one point of contact in case you need support. No dealing with different support channels, for everything inside the Synergy Platform there is just one: Renesas! Having worked a lot with different software packages from different vendors in my projects, I know what this means: Life is much easier this way, as there is no need to identify and talk to different people about the issue at hand, and eventually playing the interface between them!

Within any Synergy Software development environment, developers normally have complete on-screen visibility of all SSP source code during development and debug. However, some SSP software modules are protected, meaning that while they can be viewed on-screen and while the source will be used for compilation to achieve a small footprint, they cannot be altered, printed or saved to a file outside the development environment (see Figure 2-1). In the event that you need to go further, an SSP Source Code License available from Renesas enables the transformation of protected software modules into clear text files that can be edited, printed, and saved to a file. SSP Source Code Licenses can be purchased for individual or all SSP software modules.

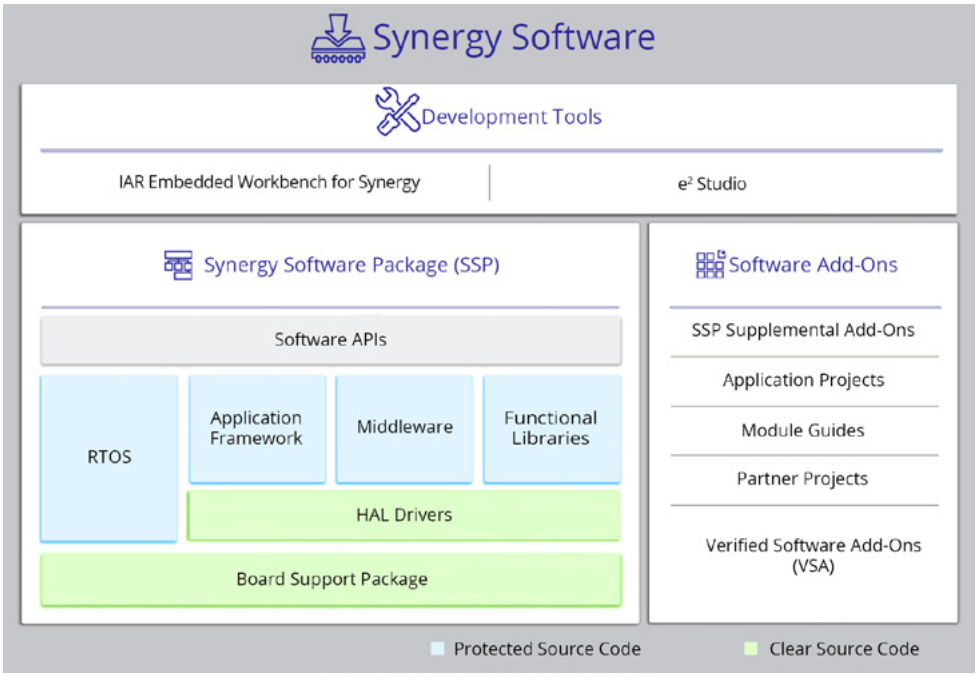


**Figure 2-1:** Parts of the source code is protected from editing, but can be unlocked by obtaining a source code license

## 2.1 Introduction to The Synergy Software Package (SSP)

As mentioned before, the Synergy Software Package (SSP) is a comprehensive piece of software covering all aspects of an embedded systems software development. It comprises the following parts:

- The **Board Support Package (BSP)**, customized for every Synergy hardware kit and Microcontroller. It includes the startup-code for all supported blocks. Developers using custom hardware can take advantage of the BSP, as it can be tailored for end products and your own board by using the Custom BSP Creator built into e² studio.
- The RTOS-independent **HAL drivers**, providing efficient drivers for all peripherals and systems services. They eliminate a lot of deep study of the underlying hardware in the microcontroller as they abstract the bit-settings and register addresses from you.
- The **Application Frameworks**, containing the system level services linking the RTOS to the Hardware Abstraction Layer (HAL) for interprocess messaging, security services, serial communication, audio playback, capacitive touch sensing, Bluetooth® low energy and much more. The completeness of these frameworks reduces errors and saves time during the development of an application.
- The **Functional Libraries** containing, for example, specialized software for digital signal processing or security and encryption related functions also reduce development time and improve the stability of the end-application.
- The **ThreadX® Real-Time Operating System**, provides a multitasking real-time kernel with preemptive scheduling and a small memory footprint. ThreadX® has been deployed in over 6.2 billion embedded devices in various areas.
- The **X-Ware™ stacks and middleware**, including file systems (FileX®), graphical user interfaces (GUIX™), USB and TCP/IP communication (USBX™, NetX™ and NetX Duo™). Everything here is completely optimized for, and integrated into, the Synergy Platform.

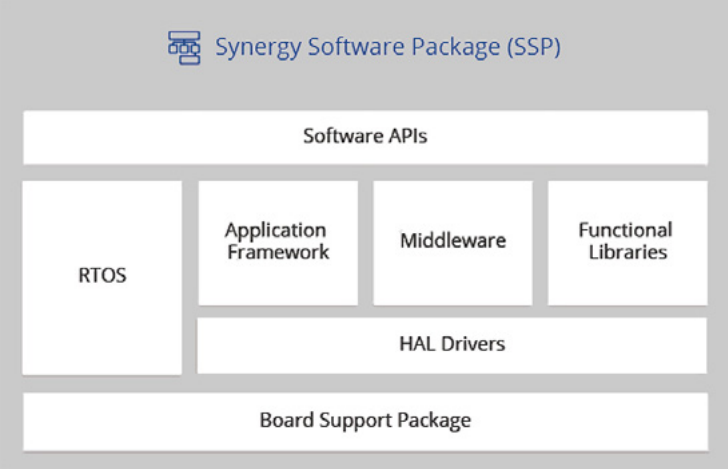


**Figure 2-2:** Everything in the Synergy software ecosystem is built in layers

Figure 2-2 shows the architecture of the different layers. Each of them can be accessed by API calls directly, or in a stacked manner. An audio application, for example, can call the audio section of the Applications Frameworks, which then uses the IIS part of the HAL to provide read / write access to an external audio converter using the serial port. Of course, an end application could also access the HAL or the BSP using API calls directly, but going through the frameworks is much more convenient.

All SSP modules support applications written in C++ and C++ applications can be configured and used in e$^2$ studio and IAR Embedded Workbench® for Renesas Synergy™.

The SSP is "production grade" software, developed using industry best practices and compliant with many MISRA C:2012 rules and guidelines, the ISO/IEC/IEEE 12207 software life cycle process and the CERT® 2nd edition. It comes with a comprehensive software datasheet stating benchmarks, code size, execution times, and the testing conditions under which these results were obtained. Renesas provides more details about their extensive quality assurance for Synergy Platform on the Synergy Solutions Gallery and in their Synergy Software Quality Handbook, available on the web as well.

Updates and bug-fixes to the SSP will happen regularly, with the scheduled releases following a roadmap and each major or minor release introducing new features. For the long term, Renesas has a great reputation for supporting their products way beyond the point at which other manufacturers would cease maintenance. With the SSP, bug fixes and enhancements are also applied to the prior major release, with bug fixes continuing to be implemented in all releases until the production end of the microcontroller, plus one year. It is lifetime maintenance, ensuring that software developers can continue working with their current baseline without having to worry about being forced to switch to a new release right away due to bugs not fixed in their version. Personally, having supported customers working on large (and long) projects for quite a while, I consider that as a great feature, as you do not need to change toolchain-versions during development. You are not left alone with software distributed on an "as is" basis.

## 2.2 Introduction to the Board Support Package (BSP)

The Board Support Package (BSP) on the bottom layer of the software architecture is a requirement for any SSP project and its responsibility is getting the MCU from reset to `main()`. Its code will set up the stacks, clocks, interrupts and the C runtime environment, before reaching the user's application. It also configures and sets up the port I/O-pins and performs any board specific initialization.

Therefore, this package is specific to a combination of a board and an MCU, which is selected during design in e² studio using the different configurators of the ISDE (Integrated Synergy Development Environment). Every development board provided is supported by a BSP. The configurators inside e² studio will extract the necessary files from the SSP and configure them based on the settings made in the user interface. The Board Support Package is heavily data driven and consists of configuration files, header files and an API.

The core of the BSP itself is compliant with the CMSIS (Arm® Cortex® Microcontroller Software Interface Standard), following the requirements and the naming conventions of that standard.

The BSP provides public functions, available to any project using the package, that allow access to the functionality that is common across the MCUs and boards supported by it. Functions include locking/unlocking the hardware and software, interrupt handling, like registering callback functions and clearing flags, software delay and register protection. The names of these functions start with `R_BSP_` and associated macros with BSP_ for an easy differentiation from other parts of the SSP. The only exception are functions providing functionality described in the CMSIS-Core. Additionally, the BSP includes a function that returns information (number and I/O-pins) about the LEDs used on a board inside a structure.

New boards and devices will be added to the BSP once they are available, assuring a solid long-term base for current and new designs. Board level support for custom boards can easily be generated using the Custom BSP Creator built into e² studio.

## 2.3 Introduction to the HAL Drivers

On top of the BSP sits the Hardware Abstraction Layer (HAL), which provides device drivers for the peripherals and which aligns with the registers of the MCU to implement easy to use interfaces, insulating the programmer from the hardware. It is a collection of modules and each of them is a driver for a peripheral available in a Synergy Microcontroller like the SPI (Serial Peripheral Interface) or the ADC (Analog-to-Digital Converter), and their names begin with `r_` for an easy differentiation from other parts of the SSP. These modules are inherently RTOS independent and are composed of two components: A low-level driver (LLD), which manipulates the registers of a peripheral directly and uses different versions of the same peripheral seamlessly and a high-level driver (HLD), whose code is specific to a Renesas hardware peripheral, but which does not access the registers directly. The HLD exposes the application programming interface to the frameworks or the user program and makes use of the LLD to interface with the microcontroller. The benefit of this architecture is that the LLD allows for very fast code and that HLD makes the APIs portable across the different Synergy MCU Series.

The interface to abstract the hardware is consistent throughout all modules of the HAL and can be extended. Some of the peripherals support multiple interfaces, while some interfaces are supported by multiple peripherals. The advantage of that is the flexibility gained, as requirements can be modified at a higher level. If, for example, the code was originally written for the dedicated hardware IIC peripheral, but later on the IIC-functionality of the SCI (Serial Communication Interface) should be used, it is sufficient to change the configuration information. The application code itself remains unchanged. While the API functionality can be accessed directly through the HAL interfaces, most of the functions can also be accessed through the different frameworks available in the SSP.

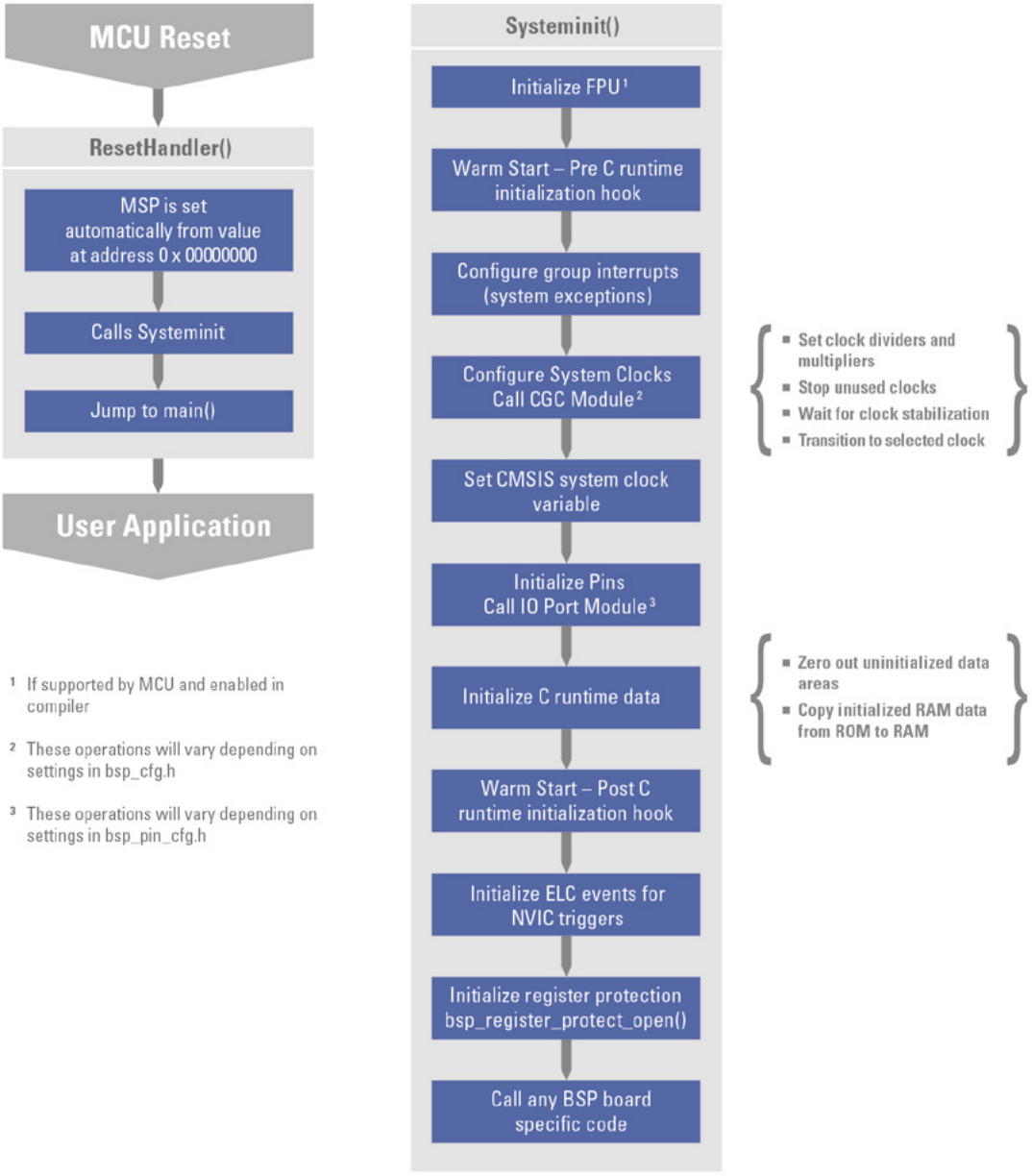


**Figure 2-3:** The flowchart for the BSP showing the move from reset to main()

## 2.4 Introduction to the Application Frameworks

Sitting on top of the HAL, the Application Frameworks provide developers with an even higher level of abstraction, allowing for better reuse of code and ease of programming, resulting in a shorter development time. The frameworks provide abstraction of various system-level and technology-specific services, enabling rich functionality with simple APIs. The different frameworks are integrated with the ThreadX® RTOS features to manage resource conflicts and synchronization between multiple user threads.

Application Frameworks are available, for example, for peripherals like SPI, ADC or IIC, for common services like audio, cap-touch sensing or JPEGs, and for communication using Bluetooth® low energy (BLE), WiFi or cellular networks. Many of the frameworks use the services of each other, and also combine several HAL and BSP calls. For example, for timing functions, the ADC framework uses services from the GPT timer interface and from the DMA or DTC for the efficient transfer of data through the shared interface.

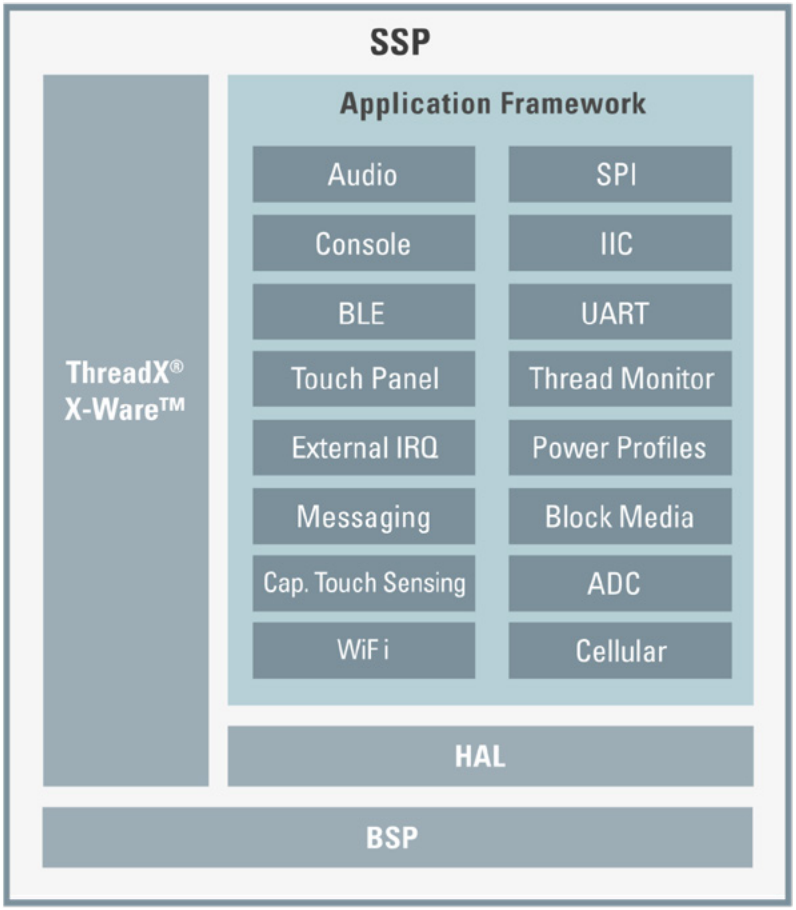


**Figure 2-4:** The Synergy Application Frameworks provide abstraction to various system-level and technology specific services

Developers using the application frameworks benefit from a higher level of abstraction, re-use of code across processors, and therefore potentially products, more consistent code, and are offloaded from the task of having to reinvent commonly used tasks across applications. At the same time, the Synergy Application Frameworks reduce the need to directly interface with the hardware available in the microcontroller even further. All module names begin with `sf_` for easy differentiation from other parts of the SSP.

## 2.5 Introduction to the Functional Libraries

Functional Libraries add more capabilities to the Synergy Platform and they interface directly with the HAL. They include extremely useful and carefully optimized functions and are callable through easy to use APIs. Included are, for example, a hardware accelerated Digital Signal Processing (DSP) library using the Arm® Cortex® Microcontroller Software Interface Standard (CMSIS), including over 60 functions for various data types, a hardware accelerated cryptography and security library with a rich set of cryptography algorithms, and a functional safety library containing software functions that operate the safety self-check features of the Synergy Microcontrollers. These software safety libraries are certified in accordance with the IEC-60730 Safety Standard for Household Appliances and IEC 61508 Safety Standard for Functional Safety.

The Functional Libraries come from Renesas and other vendors, and are pre-tested, qualified, verified and supported by Renesas, assuring the highest possible quality of the code.

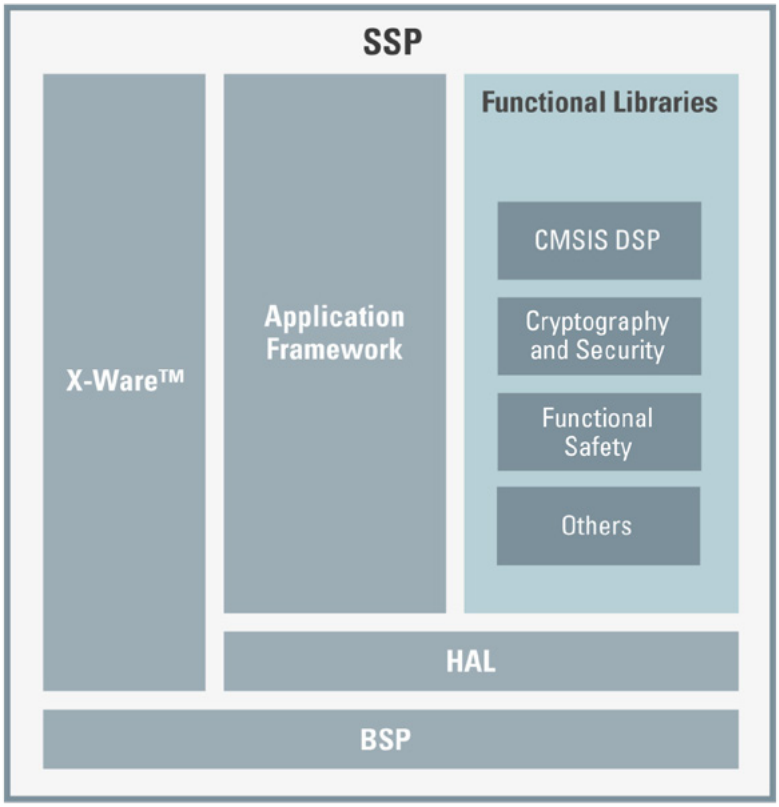


**Figure 2-5:** The Synergy Functional Libraries add additional capabilities to the Synergy Platform

Additional libraries will be added in new versions of the Synergy Software Package (SSP) over time, integrating even more features and functions, reducing the learning curve for new microcontroller developers and cutting the time-to-market for applications even further.

## 2.6 Included Middleware from Express Logic (X-Ware™)

In addition to all the software provided by Renesas, the SSP also includes additional middleware-libraries created by Express Logic, a Microsoft company, and supported by Renesas for file-handling (FileX®), Graphical User Interfaces (GUIX™), USB communication in host and device mode (USBX™) and network communication (NetX™ and NetX Duo™). These libraries, as well as the ThreadX® Real-Time Operating System, have been developed with a consistent API structure and coding style, always keeping the ease of use for the developer in mind. This unparalleled consistency makes the Synergy Software Package well equipped to speed development of new products for fast time to market.

### 2.6.1 FileX®

Express Logic FileX® is a high-performance FAT (File Allocation Table) file system for the SSP, developed especially for fast execution, fast-seek logic and small footprint. It is fully integrated with the SSP and ThreadX® and is available for all supported Synergy MCUs.

Supported are all FAT formats, including FAT12, FAT16, FAT32 and exFAT (extended FAT). FileX® is built especially for real-time embedded systems, maintaining an internal FAT entry cache. Other features are:

- Contiguous file allocation
- Consecutive sector and cluster read / write
- Long filenames and 8.3 naming conventions
- Unlimited creation of FileX® objects like media, directories and files
- Optional fault tolerance

FileX® supports USB mass storage devices, SD / eMMC cards, SPI flash, QSPI flash and on-chip flash, as well as RAM as storage media.

A high reliability is achieved through error detection and recovery capabilities, different fault tolerance options and built-in performance statistics.

The library interfaces to the other parts of the SSP through the Block Media Interface (`fx_io`), an abstract interface using function pointers instead of direct function calls, acting as adaption layer for the block media device drivers. Functions are called between file systems and the Synergy Platform block media drivers in the Application Frameworks, such as SD/MMC and SPI Flash, which in turn depend on the SDMMC, Flash and SPI interfaces of the HAL.

The interface remains the same for any media driver. All media drivers appear functionally identical at the file I/O layer and can be interchanged with one another without changing application code.

### 2.6.2 GUIX™

Express Logic GUIX™ runtime library is the SSP's high-performance Graphical User Interface framework that is optimized for the Synergy MCU architecture and enables the creation of elegant user interfaces. It is fully integrated with the ThreadX® RTOS and was designed to have a small footprint. It is implemented as a pure C library and only brings features used by the application into the final image. Due to a minimal function call layering and optimized clipping, drawing, event handling, and support for the Renesas hardware graphics acceleration, the response is quite fast.

The runtime library of GUIX™ supports multiple screens and multiple languages with UTF-8 string encoding. It includes functionality for horizontal, vertical and drop down lists, single- or multi-line text for either view or input, checkboxes, buttons, sliders, and scrollbars. The supporting framework enables event queues and signals, windowing and viewport management, as well as clipping, deferred drawing and dirty list maintenance.

GUIX™ has a dedicated desktop design application called GUIX Studio™, which provides a complete WYSIWYG screen design environment, where developers can drag-and-drop graphical elements to build compelling user interfaces, which can be executed on a Windows® workstation within GUIX Studio™, allowing quick and easy demonstration and evaluation of user interface concepts.

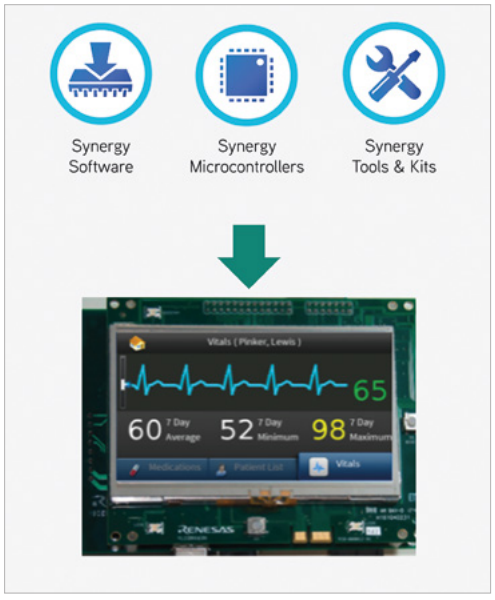


**Figure 2-6**: GUIX™ allows the easy creation of compelling GUIs

Once the UI is finalized, GUIX Studio™ will generate C code compatible with the GUIX™ library, ready to be compiled. Developers can produce pre-rendered fonts for use within an application, and fonts can be generated in monochrome or anti-aliased formats using the integrated font generation tool that supports any set of characters, including Unicode characters for multi-lingual applications. GUIX Studio™ allows customization of the default colors and drawing styles of GUIX™ widgets, allowing developers to tune the appearance of their application to their needs.

As always, functionality of the runtime library gets called through intuitive, readable and highly functional API calls. Building applications based on GUIX™ is easy: include the `gx_api.h` file in the application source code and link the objects with the GUIX™ library.

### 2.6.3 USBX™

USBX™ is Express Logic's small footprint and high-performance USB host and device stack for SSP applications, allowing a Synergy MCU to interface with USB devices or to be connected with Windows®, Apple® and Linux workstations over USB. It is fully integrated with ThreadX® and available for all supported Synergy MCUs.

The stack includes support for the Low Speed (1.5 Mbps), Full Speed (12 Mbps) and High Speed (480 Mbps) transfer modes and the USB 1.1 and USB 2.0 specifications. It can act in two modes: host and device.

The host mode of USBX™ is used when the application requires communication with certain USB devices such as a USB keyboard, a USB printer or USB Flash disk, this means with peripherals acting as slaves. There are two major components responsible for this mode: The USB core stack and the USB controller.

The USBX™ core stack is responsible for detection of device insertion and removal, as well as all the protocols available to USB (control, bulk, interrupt and isochronous). The core stack ensures correct device detection and configuration, and its plug-and-play mechanism searches for a USB class driver responsible for this device.

The USB controller supports major USB standards like OHCI and EHCI. It is possible for a single host system to have multiple host controllers. The USB class driver(s) are responsible for driving each USB device once it has been enumerated. Several USB standard classes are supported, such as CDC, HID, HUB, STORAGE, AUDIO or PRINTER, but proprietary classes can be used as well. USBX™ host mode can support cascading hubs, multiple configuration devices, and composite devices.

The device mode of USBX™ is used when the application should communicate with a Windows®, Apple® or Linux desktop. In this case the Synergy MCU is considered to be a USB device or slave. The architecture for the device side is similar to the host side although simplified. The USBX™ device core stack is responsible for handling the USB host enumeration requests. The USB device driver class is responsible for driving the USB device once enumeration has taken place by the host. In the device mode, USBX™ can support a complex device framework including multiple configurations, multiple interfaces, and multiple alternate settings. As with the USB host mode, support for several of the standard USB classes like CDC, HID, STORAGE, PIMA or RNDIS is included in the library.

Working with the USBX™ library is very straightforward and a first working version can be accomplished in a very short timeframe. All what needs to be done is to add and configure a thread, a queue and four modules in the Synergy configurator in e$^2$ studio. In the exercise provided in chapter 10, you will experience this simplicity yourself.

### 2.6.4 NetX™ / NetX Duo™

Express Logic's NetX™ and NetX Duo™ implement a TCP/IP protocol application for the SSP. While NetX™ provides a streamlined IPv4 capable TCP/IP stack, NetX Duo™ provides IPv4 and IPv6 capabilities in a dual-stack product, offering a future proof solution.

NetX™ and NetX Duo™ have a unique Piconet™ architecture, where only those services and protocols that are actually used are included in the image. Combined with a zero-copy API, which eliminates the processor cycles normally consumed by moving data to and from user buffers, they are perfect for applications that require network connectivity, as applications can use the freed processor cycles for more useful tasks. NetX Duo™ conforms to RFC standards, verified by the industry standard Ixia IxANVL™ test suite, and has achieved IPv6-Ready Logo certification, evidence that it has passed conformance and interoperability tests, administered and validated by the IPv6 Forum.

In addition to the zero-copy API, NetX™ also provides a BSD socket-compatible API for applications with legacy BSD application code. The application may create any number of packet pools in any number of memory areas. These zero-copy packets can be linked with packets from the same pool or even a different pool to accommodate larger payloads.

Basic UDP packets pass through NetX™ and NetX Duo™ without any copying or system context switches; they are delivered directly to waiting threads. This is referred to as UDP Fast Path™ Technology.

NetX™ and NetX Duo™ include network configuration protocols like DHCP (client / server), SNMP, and SNTP (network time protocol), domain name services like DNS, and NAT and email transfer with POP3 and SMTP. HTTP web server / network management is available, as are the connectivity protocols PPP, Telnet, FTP and TFTP. NetX Duo™ has received safety certifications according to IEC 61508 SIL 4, IEC 62304 Class C, ISO 26262 ASIL D and EN 50128 SW-SIL4.

Additional components like HTTPv6, FTPv6, DNSv6, Telnetv6, DHCPv6, MQTT and NetX™ Secure (TLS) are available for NetX Duo™. Higher level solutions such as SSL/SSH/TLC are available as Synergy Verified Software Add-on (VSA) components. Information about them can be found in the Synergy Solutions Gallery on the Web.

## 2.7 The RTOS of Choice: ThreadX®

ThreadX®, created by Express Logic, is the Real Time Operating System (RTOS) of choice for the Synergy MCU Family. It was developed specifically for both high-end and graphic rich applications, as well as for embedded systems with limited memory and special requirements in terms of determinism. It features a small Flash-memory footprint (less than 2 KB on an S3, S5 or S7 MCU), a small RAM requirement (minimum of < 1 KB for kernel RAM) and a short context switching time of 0.7 µs on a Synergy R7FS7G2 MCU.

As a multitasking RTOS, ThreadX® uses multiple advanced scheduling algorithms, provides real-time event trace, message passing and interrupt management as well as many other services and is fully deterministic. It has a proven reliability record with more than 6.2 billion deployments in the consumer, medical electronics and industrial automation markets, and meets numerous important safety and quality standards.

Other advanced features offered are for example its picokernel™ architecture, Preemption-Threshold™ scheduling, Event-Chaining™ and a rich set of system services. It also supports the X-Ware™ middleware with the services required by the different components.

### 2.7.1 Why Use an RTOS?

Not so long ago, the life of a software-developer for an embedded system was comparably straightforward. Most systems consisted of a simple main loop, where a single background task was running in an endless loop. It executed tasks like reading inputs from I/O-pins (for example the state of a push-button), performing certain calculations like multiplying a measured voltage and a current to obtain a power consumption, updating discrete outputs and so on.

The few peripherals connected to the microcontroller issued an interrupt to notify the CPU that maybe a new conversion result from a data converter is be available and should be read, or that the RS-232 port finished transmitting the last word and is waiting for new data. These asynchronous events have been traditionally handled inside the interrupt service routines. And if no interrupt was happening, the software was idling in the background loop, waiting for one to occur.

With embedded systems growing more complex, having a higher connectivity to the outside world, or more user interfaces to serve and more tasks to execute, the setup mentioned above is hard to maintain, and an operating system is becoming not only something preferable, but a necessity. Why? The response time of the background loop was hard to predict and definitely not deterministic, as it was affected by the decision tree inside the code and would even change once a modification to the software was made. Another disadvantage was that all tasks (or "threads") in the background-loop had the same priority, as the code was executed serially, which means that the reading of the push-button and the power-calculation in the example above would always have been executed one after another, even if the button was pressed multiple times. In other words, some of these events could have been missed, something which is a no-go in embedded computing. The system needs to be more responsive and more predictable; it needs to be real-time.

Having seen the clear need for an operating system, the next question is: Why not use an off-the-shelf OS like Windows® or Linux? They are part of our everyday life and seem to do their job quite well. There are a couple of reasons for not using them. First of all, they offer by far too many features, not needed in an embedded system, and they are not configurable enough. Secondly, they need more resources and memory space than what is available in a resource-restricted embedded application. And last but not least, their timing uncertainty is still too large to fulfill real-time requirements.

All of this means that there are special requirements for an OS running on our embedded systems. The first and most important one is the predictability of the timing behavior. An RTOS needs to be deterministic, which means that the upper limits of the blocking times, like the time during which interrupts are disabled, need to be known and available to the developer.

The second requirement is that the RTOS has to manage the timing and scheduling. It has to be aware of the timing constraints and deadlines and it must provide a time service with a high resolution. And finally, the operating system should be fast, small and configurable.

Another major function provided by an RTOS is thread management, allowing the execution of quasi-parallel tasks on an MCU using threads (a thread can be understood as a lightweight process) by taking care of the state of the threads, the queuing of the processes, and allowing for preemptive threads and interrupt handling. Other services provided to the application are scheduling, thread synchronization, inter-thread communication, resource sharing, and a real-time clock as time reference.

While many software designers balk at using an RTOS, because they fear its complexity and learning curve, real-time operating systems offer plenty of benefits like the increased responsiveness to real-time events, the prioritization and easy addition of threads, the reduced development time once the first steps are mastered, and services added to the application.

As with all good things in life, there are of course some disadvantages in using an RTOS: It needs additional RAM and Flash memory and each thread needs an additional stack, increasing the memory need even further. Cost is another factor, but this does not count in our special case: When using Synergy Platform, all is included. And the simple-to-use API of ThreadX® as part of the SSP reduces the steepness of the learning curve a lot!

### 2.7.2 The Main Features of ThreadX®

Having seen the benefits of using an RTOS, it is time now to look into the main features of ThreadX®. It is small in size, with a very small RAM-memory footprint and is optimized for very fast performance and low overhead, leaving the resources of the microcontroller mostly to the application.

The enhanced real-time scheduling algorithms and efficient multitasking routines provide round-robin scheduling and time slicing, as well as preemptive and Preemption-Threshold™ scheduling. To prioritize threads, there are up to 1024 priority levels available to the software-engineer, with the default number of priorities being 32.

Resources in an embedded system are mainly limited by time and memory resources. ThreadX® provides several powerful options to manage them. Memory needs to be allocated fast and in a deterministic manner, so the SSP provides the ability to create and manage any number of pools of fixed-sized memory blocks. As the pools are fixed size, memory fragmentation is not a problem, but the size of a memory block must be the same or larger than the largest memory request from the application or the allocation will fail. But making it large enough for that scenario might waste memory if requests come in with different block sizes. A workaround for that is the creation of several memory block pools that contain different sized memory blocks. This disadvantage is outweighed by the fast allocation and de-allocation of these blocks, as these allocations are done at the head of the available list. This provides the fastest possible linked list processing and might keep the actual memory block in cache.

ThreadX® does not only allow block pools with fixed block sizes, but also the creation of multiple byte pools, which behave similar to a standard C heap. Memory is only allocated with the desired size in bytes, based in a first-fit manner. The disadvantage of that is that, like heaps in C, byte pools get easily fragmented, creating a somewhat un-deterministic behavior.

Other resources managed by the RTOS are the application timers, allowing an unlimited number of software timers to be created. These application timers are available in three operation modes: one-shot, periodic and relative. A one-shot timer will call a user-function only once after the timer expires, while a periodic timer calls a user-function repeatedly after a fixed interval. The relative timer is a single continuously incrementing 32-bit tick counter. All the timer expirations are specified in terms of ticks, for example, 1 tick equates to 10 ms, but this is of course configurable. The SSP manages activation and expiration without linear searching, reducing the amount of overhead in timer-centric applications like communications and process control.

Synchronization of threads and communication between tasks is another big topic in embedded systems. ThreadX® provides several mechanisms for that and makes this task very easy and straightforward. Semaphores and mutual exclusions (mutex) help preventing priority inversion, and allow an unlimited number of objects to be created. Sophisticated callback functions and Event-chaining™ are available as well, as is optional priority inheritance. In addition, ThreadX® can suspend in either FIFO or priority order, avoiding the problem of threads starving for processing time.

Event flags are another thread synchronization feature. An event flag group consists of 32-bits, where each bit represents a different logical event and threads can wait on a subset of these bits. Event flags also support Event-chaining™. As with semaphores, an unlimited number of objects can be created and information on the run-time performance of the flags is available.

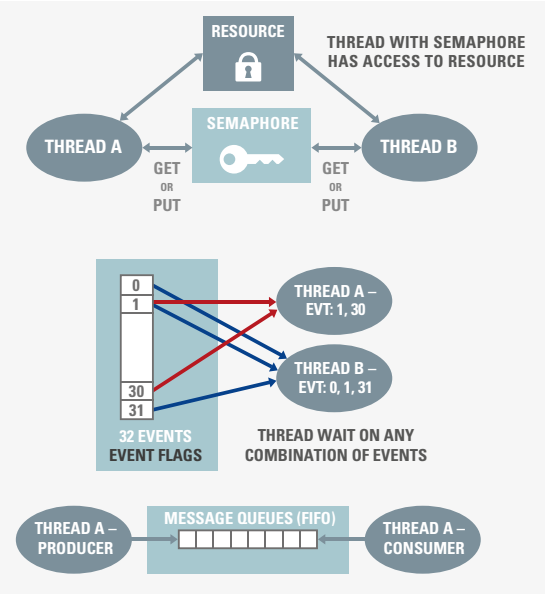


**Figure 2-7:** ThreadX® synchronization and communication features

The last synchronization feature to be mentioned here is message queues. Using the producer-consumer model for inter-task communication, ThreadX® supports messages sizes from 1 to 16 four-bytes words, with mailboxes being a special case of a message queue with the size 1. Application threads can register a callback function for further notifications. As with the event flags, run-time performance information is available.

For debugging purposes, a very important feature is implemented: A built-in event trace capability supported by TraceX®, which allows to view the sequence of the different events. TraceX® is a standalone Windows® application, but can be launched from inside e² studio.

And finally, ThreadX® is compliant with Misra-C:2004, Misra-C2012, EAL4+ Common Criteria security certified, and is pre-certified to the following standards:

- IEC 61508 SIL 4
- IEC 62304 Class C
- UL/IEC 60730-1 H
- CSA E60730-1 H
- ISO 26262 ASIL D
- UL/IEC 60335-1 R
- UL 1998
- EN 50128 SW-SIL 4

**Points to take away from this chapter:**

- The Synergy Software Package (SSP) is built in layers and all of them can be accessed by simple API-calls.
- The Application Frameworks remove the need to write basic code.
- ThreadX® and the X-Ware™ middleware are well proven components making the creation of full-featured applications a snap.