GUIX Studio (1)

Każda aplikacja wymagająca interakcji z użytkownikiem musi być wyposażona w interfejs użytkownika. Współcześnie zwykle do tego celu używa się wyświetlaczy graficznych. Zastosowanie takiego wyświetlacza znacznie podnosi atrakcyjność urządzenia i umożliwia – w myśl maksymy „1 obraz = 1000 słów" – stosowanie symboli graficznych, zamiast opisów tekstowych. Jednak zaprojektowanie interfejsu użytkownika z wyświetlaczem graficznym jest dość trudne i dlatego producenci podzespołów dostarczają odpowiednich narzędzi ułatwiających pracę programisty lub konstruktora systemu. Przykładem jest GUIX Studio firmy Renesas.

Moduł ewaluacyjny SK-S7G2 (**fotografia 1**) jest przeznaczony do prototypowania i testowania aplikacji IoT. Wyposażono go w kolorowy wyświetlacz graficzny TFT 2,4" o rozdzielczości 240×320 pikseli. Panel wyświetlacza ma wbudowany sterownik ILI9341V produkowany przez firmę ILTEK. Komunikacja pomiędzy sterownikiem a mikrokontrolerem odbywa się przez interfejs równoległy pracujący w standardzie przemysłowym Intel 8080 lub przez SPI. Magistrala może mieć szerokość 8, 9, 16 lub 18 bitów.

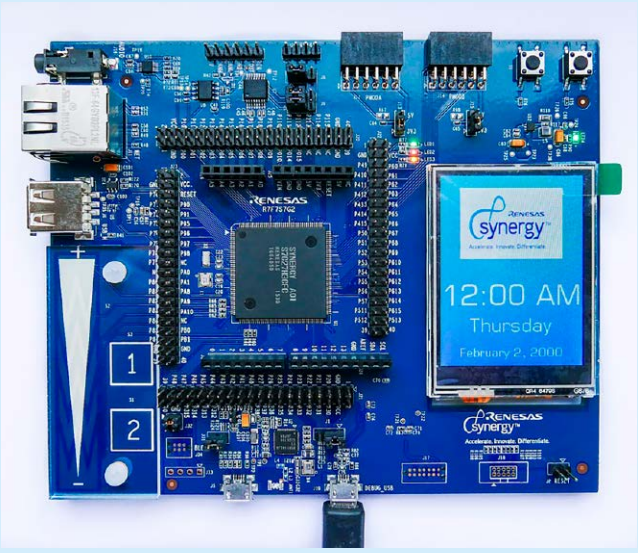
W module SK-S7G2 użyto 16-bitowej magistrali równoległej zajmującej 27 linii portów mikrokontrolera. Moduł wyświetlacza jest połączony z płytką za pomocą taśmy elastycznej i specjalnego złącza. Jeżeli linie sterujące są potrzebne do innych celów, to taśmę można odłączyć.

Na ekranie LCD zamontowano dotykowy panel rezystancyjny ze sterownikiem SX8656 produkowanym przez firmę Semtech. Zmiany rezystancji są mierzone metodą techniczną przez przetworniki A/C o rozdzielczości 12 bitów. Sterownik komunikuje się z mikrokontrolerem za pomocą interfejsu I²C.

Oprogramowanie wyświetlacza graficznego jest pracochłonne. Aby zaoszczędzić czas potrzebny na napisanie na przykład procedur rysowania elementów, wykorzystuje się gotowe biblioteki prekompilowane lub z kodami źródłowymi albo aplikacje generujące procedury na podstawie graficznego projektu ekranu (ekranów) wyświetlacza.

W artykule opisano metodę, jak krok po kroku zaprojektować i zaprogramować ekran wyświetlający informacje na wyświetlaczu modułu. Do tego celu będziemy potrzebowali:

- Środowiska projektowego Synergy e2studio.
- Biblioteki Synergy Software Platform SSP.
- Aplikacji GUIX Studio.

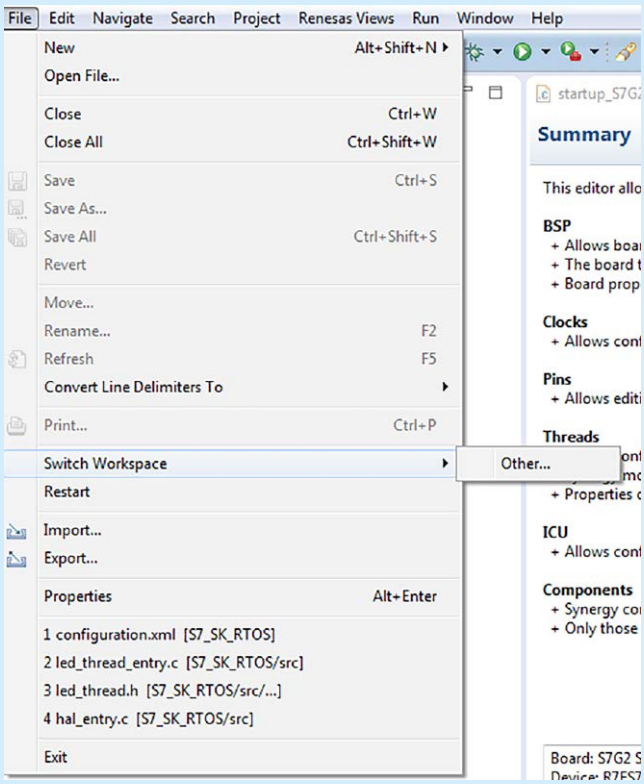


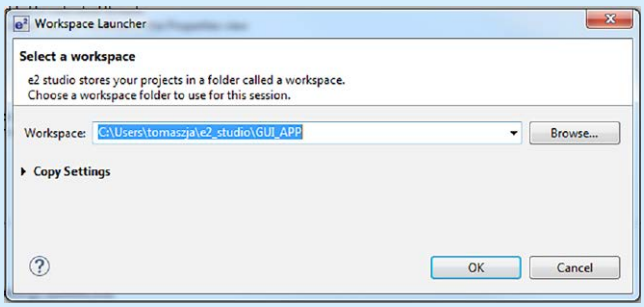
Fotografia 1. Moduł SK-S7G2 Starter Kit z wyświetlaczem graficznym

Wymienione programy są dostępne po zarejestrowaniu się na stronie firmy Renesas.

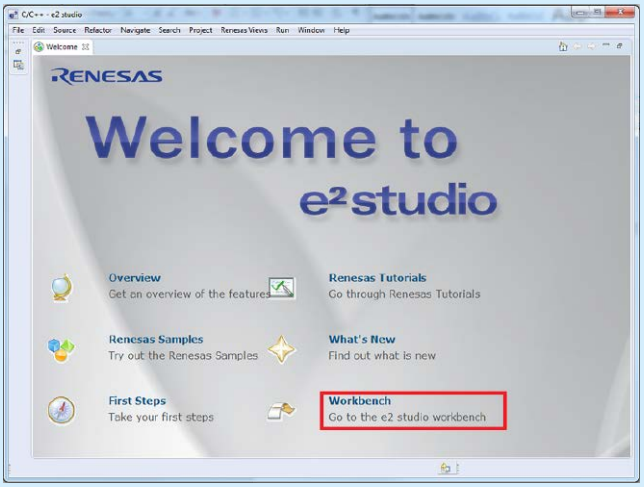
Projekt e2studio

Pracę zaczynamy od utworzenia i wstępnego skonfigurowania nowego projektu w środowisku e2studio. W tym momencie zakładamy, że e2studio zostało wcześniej skonfigurowane do wspierania projektów utworzonych za pomocą Synergy i że zainstalowano kompilator GCC ARM Embeded z plikiem licencji. Ponadto ze strony Renesasa trzeba pobrać i zainstalować pakiet Synergy Software Package SSP. Te czynności zostały opisane w artykule na temat e2studio.

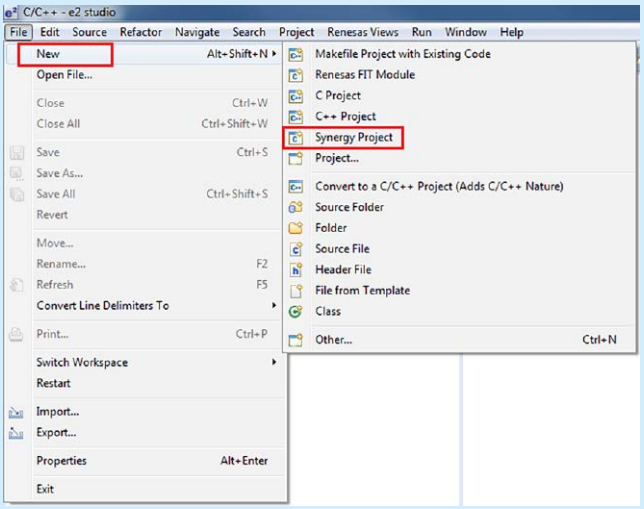




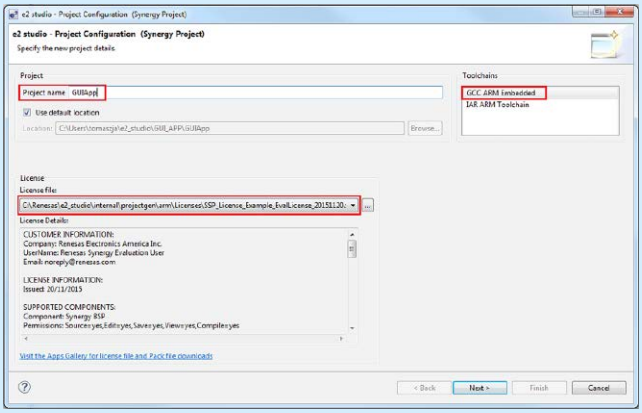
Rysunek 2. Tworzenie nowego workspace'u

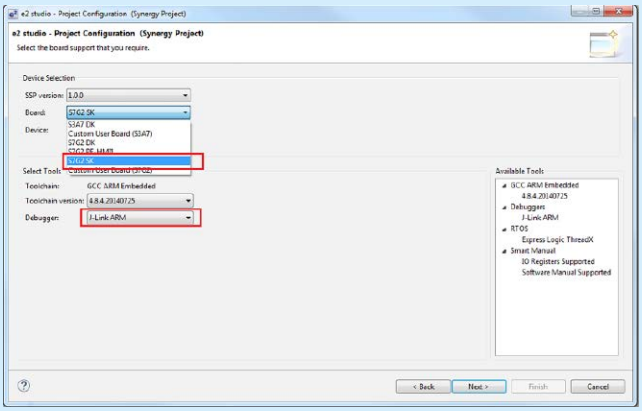


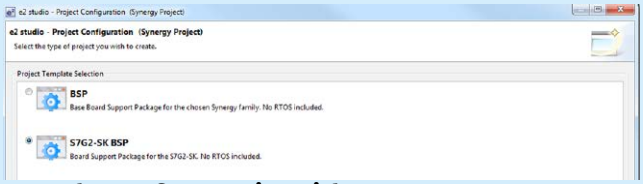
Rysunek 3. Tworzenie nowego środowiska



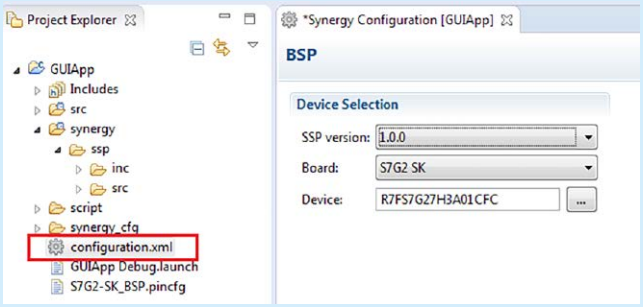
Rysunek 4. Tworzenie nowego projektu Synergy







Rysunek 5. Konfigurowanie projektu



Rysunek 6. Szablon projektu

Najpierw tworzymy nowe środowisko projektowe (workspace) File → Switch Workspace → Other (**rysunek 2**). W oknie *Workspace Launcher* nadajemy unikalna nazwę – proponuję, aby teraz było to **GUI_APP**. Po kliknięciu na OK otworzy się okno pokazane na **rysunku 3**. Wybieramy opcję *Workbench*, po czym zostaje wyświetlone okno e2studio.

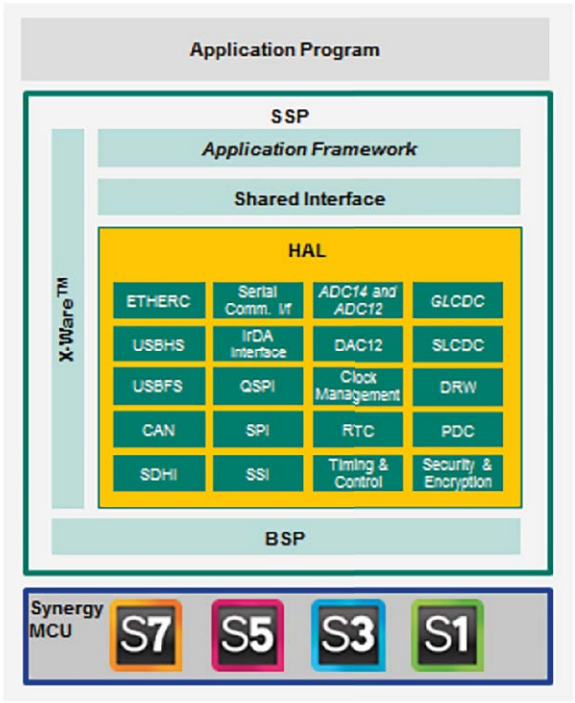
Teraz pora na nowy projekt. E2studio tworzy nowy projekt po wybraniu File → New → Synergy project (**rysunek 4**).

Nowy projekt musi być spersonalizowany: trzeba mu nadać nazwę – niech to będzie **GUIApp**. Następnie należy wybrać kompilator (GCC ARM Embeded) oraz podać ścieżkę dostępu do jego pliku licencji (jeśli nie zrobiono tego wcześniej). W kolejnym kroku z listy wspieranych modułów ewaluacyjnych wybieramy **S7G2 SK** oraz debugger **J-Link ARM**. W konfiguracji wybieramy opcję S7G2-SK BSP (**rysunek 5**) Konfigurator utworzy szablon projektu pokazany na **rysunku 6**.

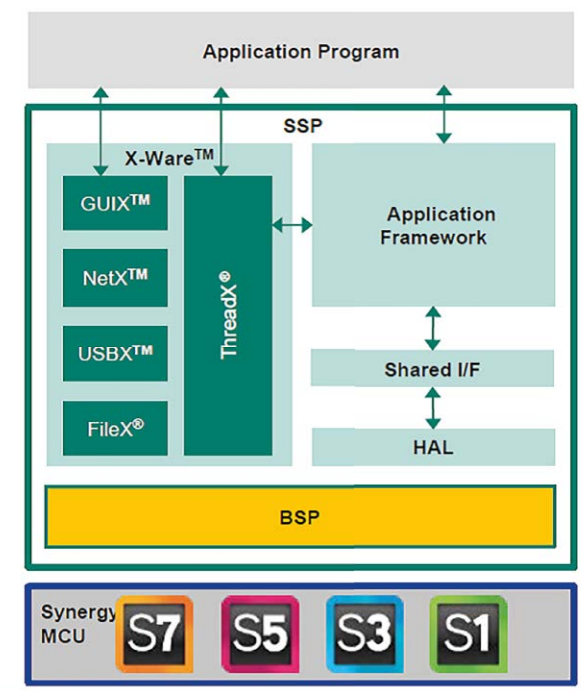
W tym momencie możemy przystąpić do konfigurowania projektu nazwanego przez nas GUIApp. Robi się to w oknie Synergy Configuration GUIApp. Zostaje ono otwarte automatycznie po wygenerowaniu szablonu, ale można je też wyświetlić, klikając na plik *configuration.xml* umieszczony w oknie projektu (rys. 6) lub na przycisk *Synergy Configuration*. Okno konfiguracji ma kilka zakładek: *Summary*, *BSP*, *Clock*, *Pins* itd. Konfiguracja jest dość skomplikowana i będzie się odbywała w wielu krokach. Musimy skonfigurować cały driver sterowania wyświetlaczem, począwszy od warstwy najwyższej, a skończywszy na warstwie sprzętowej. Obsługa wyświetlacza będzie realizowana pod kontrolą RTOS ThreadX.

Nasza aplikacja ma budowę warstwową. Najwyższa warstwa aplikacji będzie zawierała procedury wygenerowane przez aplikację GUIX Studio oraz procedury użytkownika odpowiedzialne za synchronizację działania całej aplikacji. Kolejna, niższa warstwa, to warstwa HAL (Hardware Abstraction Layer) zawierająca procedury – drivery przeznaczone do obsługi bloków funkcjonalnych wbudowanych w mikrokontrolery Synergy. Jednak HAL jest warstwą niezależną od szczegółowych rozwiązań sprzętowych (hardware), różnych dla różnych typów mikrokontrolerów. Ta niezależność pozwala na użycie takich samych procedur, na przykład obsługi timerów wszystkich typów mikrokontrolerów. Na **ry unku 7** pokazano schematycznie warstwę HAL w otoczeniu warstw wyższych i niższej BSP.

Najniższa warstwa BSP (Board Support Package) zapewnia współpracę aplikacji z konkretnym typem mikrokontrolera i jej



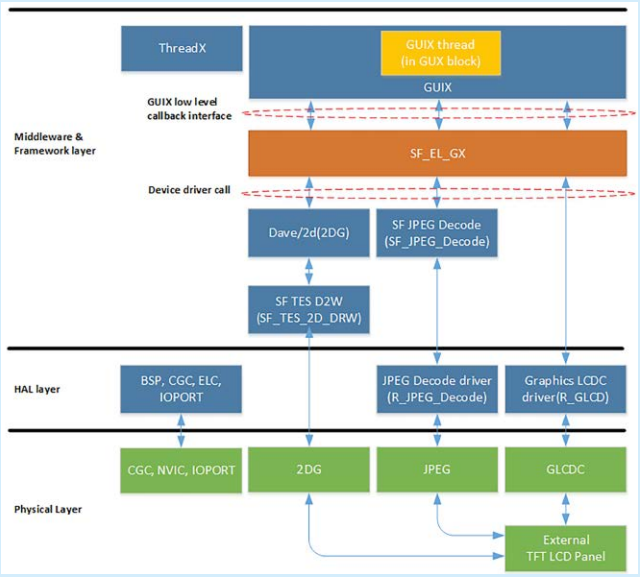
Rysunek 7. Warstwa HAL



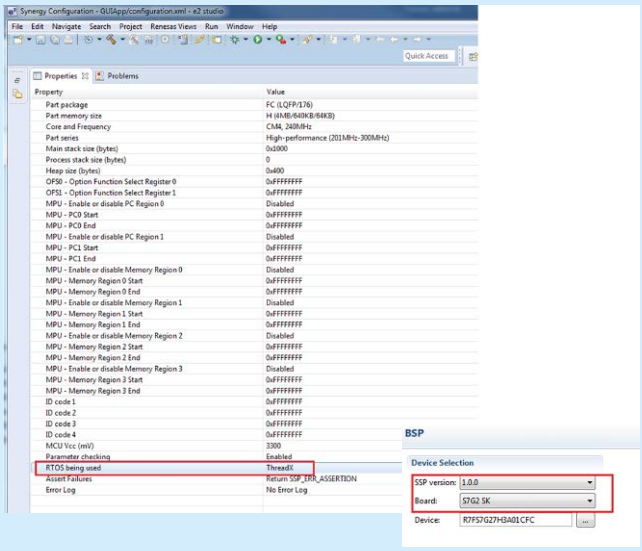
Rysunek 8. Umieszczenie warstwy BSP w bibliotece SSP

procedury muszą być napisane dla każdego z nich osobno. SSP zawiera warstwę BSP dla modułów ewaluacyjnych DK-S7G2, PE-HMI1 i SK-S7G2. Na **rysunku 8** pokazano budowę modułową biblioteki SSP z umiejscowieniem warstwy BSP. SSP dostarcza gotowe rozwiązania do obsługi biblioteki graficznej, interfejsu USB, rozwiązań sieciowych i systemu plików oraz własny system wielozadaniowy RTOS ThreadX.

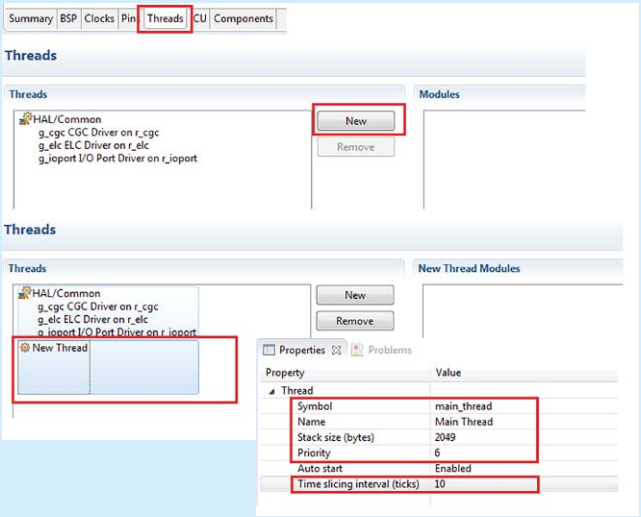
Projekt interfejsu będzie się opierał na module portu GUIX Synergy umieszczonego w warstwie framework biblioteki SSP. Moduł portu i jego współpraca z warstwami niższymi HAL i warstwą fizyczną został schematycznie pokazany na rysunku 9. Port umożliwia współpracę aplikacji Express Logic GUIX z warstwami Framework i niższymi.



Rysunek 9. Moduł portu GUIX Synergy

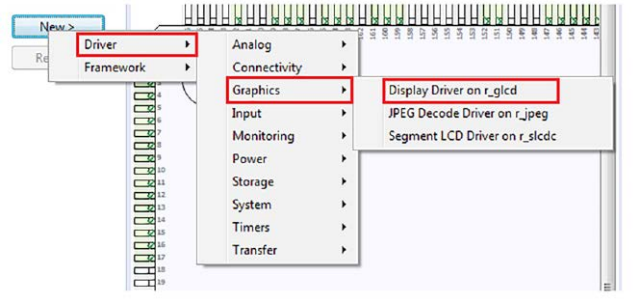


Rysunek 10. Zakładka konfiguracji BSP i wybór RTOS ThreadX

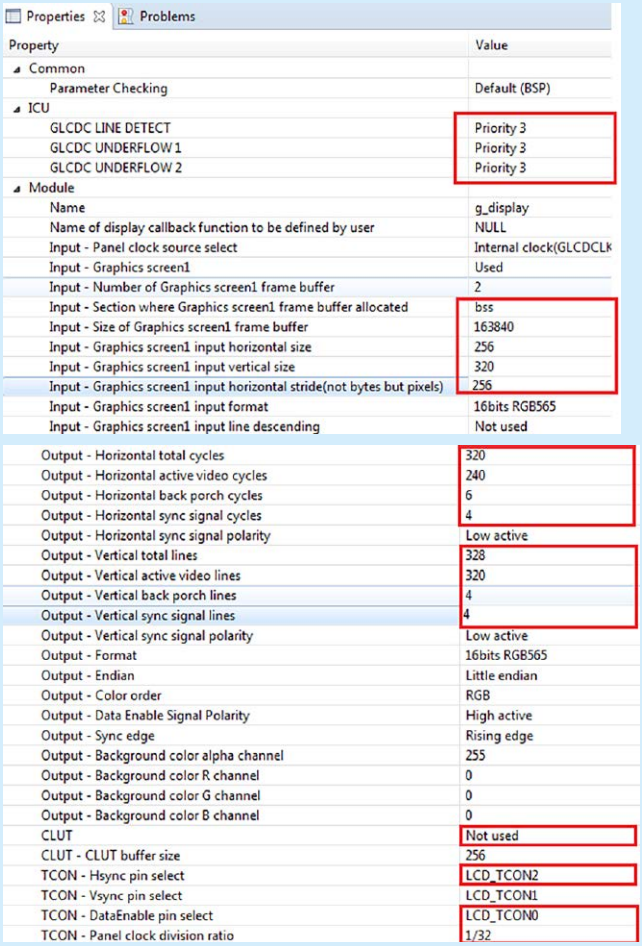


Rysunek 11. Dodanie nowego wątku

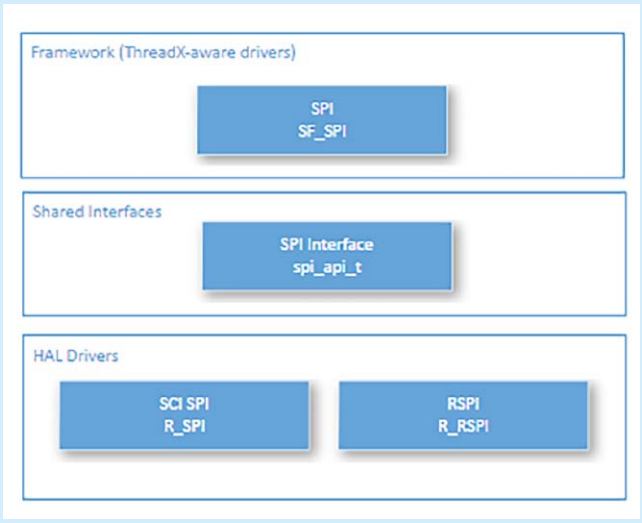
Konfigurowanie projektu zaczynamy od zakładki BSP i jej okna *Properties*. Wybieramy tu wersję biblioteki (w momencie pisania artykuły była dostępna tylko wersja 1.1.0) i typu wspieranego modułu ewaluacyjnego. W wierszu „RTOS being used" wybieramy opcję



Rysunek 12. Dodanie drivera wyświetlacza graficznego

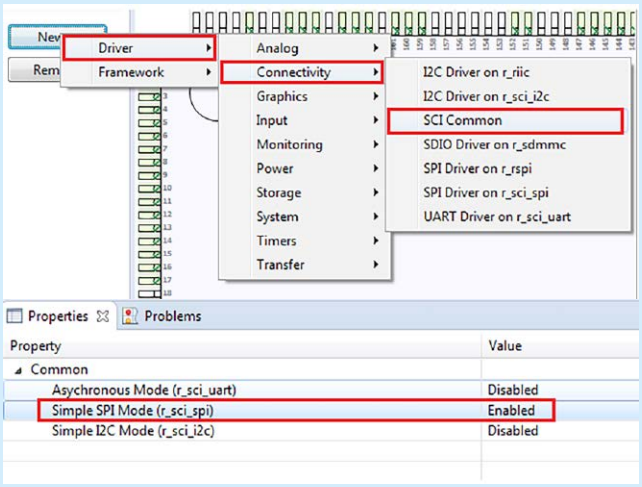


Rysunek 13. Konfigurowanie drivera graficznego w oknie *Properties*

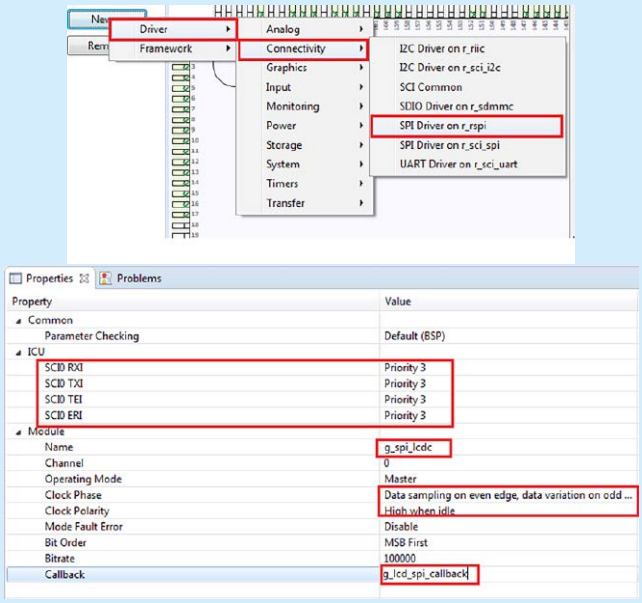


Rysunek 14. SPI framework

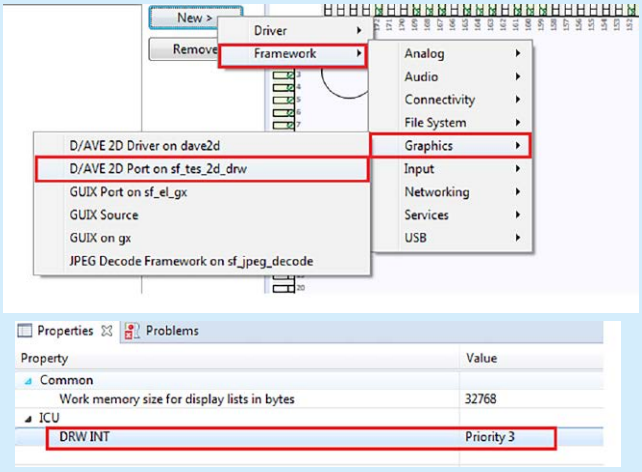
ThreadX (**rysunek 10**). Po wybraniu używanego RTOS dodajemy nowy wątek o nazwie Main Trherad. W tym celu w oknie konfiguracji wybieramy zakładkę *Threads*, potem klikamy na przycisk *New* – zostanie utworzony nowy wątek *New Thread*. We właściwościach



Rysunek 15. Dodanie i konfigurowanie drivera SCI Common (warstwa HAL)



Rysunek 16. Dodanie i konfigurowanie drivera SPI Driver on r_rspi (warstwa HAL)



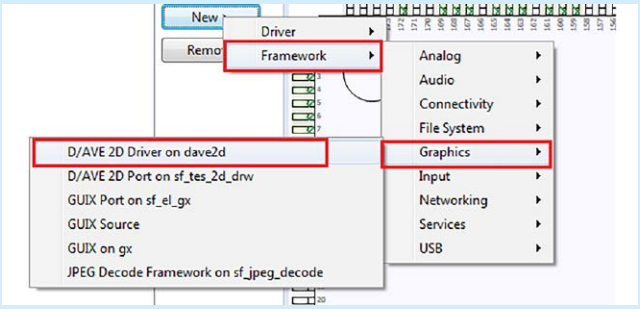
Rysunek 17. Dodanie i konfigurowanie D/AVE 2D Port on sf_tes_2d_drw

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **39483**, pass: **5kc7a2ku**

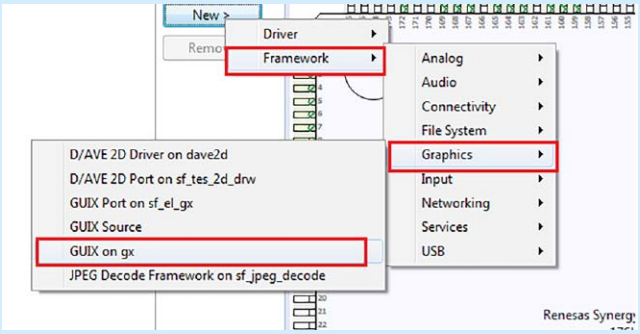
wątku trzeba zmienić wartości domyślne: *Symbol = main_thread, Name = Main Trhread*, wielkość stosu = 2048 bajtów, priorytet = *6*, interwał czasowy = *10 tick* (**rysunek 11**).

Do tak utworzonego wątku w następnym kroku dodajemy moduł drivera wyświetlacza LCD (warstwa HAL). W tym celu w oknie *Main Threads Modules* klikamy na *New* i kolejno wybieramy Driver → Graphics → Display Driver on r_gldc (**rysunek 12**). Następnie dodany driver jest konfigurowany w oknie *Properties*, jak na **rysunku 13**.

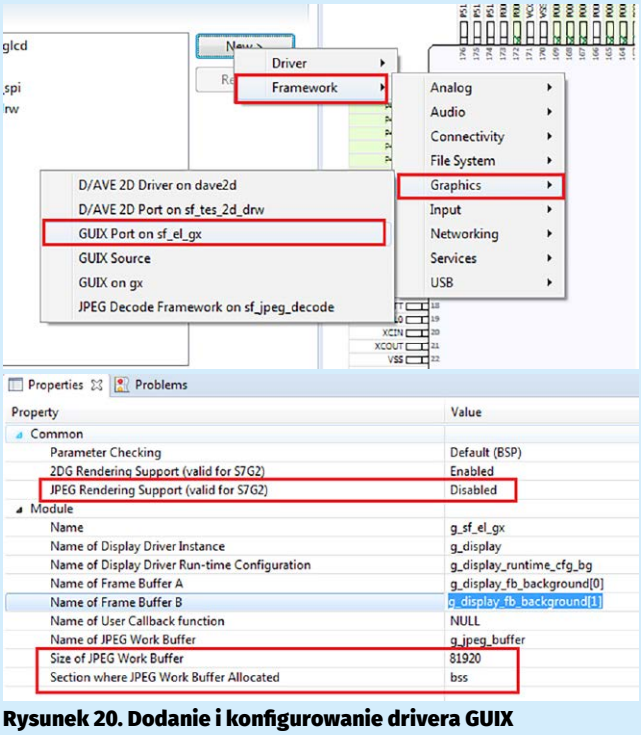
Użycie systemu RTOS powoduje, że przesyłanie danych z/do sterownika wyświetlacza wymaga implementacji specjalnych funkcji komunikacyjnych. W tym wypadku będą to drivery i funkcje API interfejsu SPI. Strukturę interfejsu *SPI Framework* pokazano na **rysunku 14**. Jako pierwszy dodajemy driver komunikacji sterownika wyświetlacza z mikrokontrolerem (warstwa HAL). Jak poprzednio,



Rysunek 18. Dodanie frameworka D/AVE 2D Driver on dave2d



Rysunek 19. Dodanie funkcji frameworka GUIX

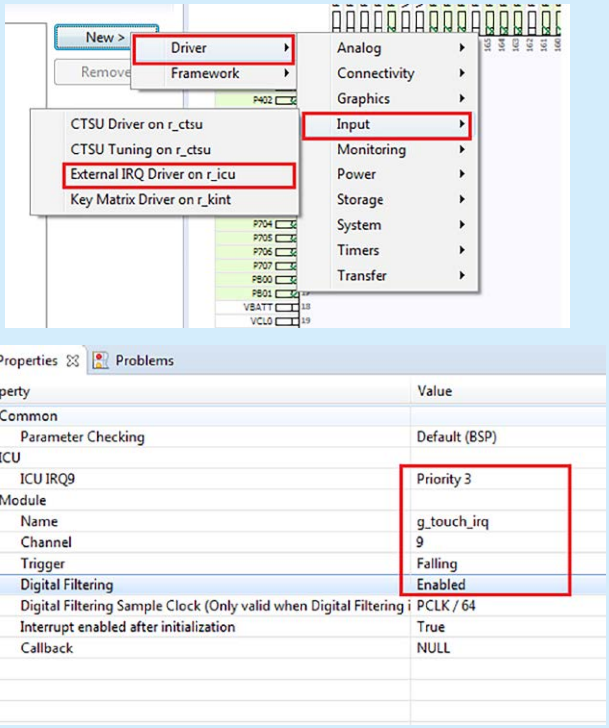


Rysunek 20. Dodanie i konfigurowanie drivera GUIX

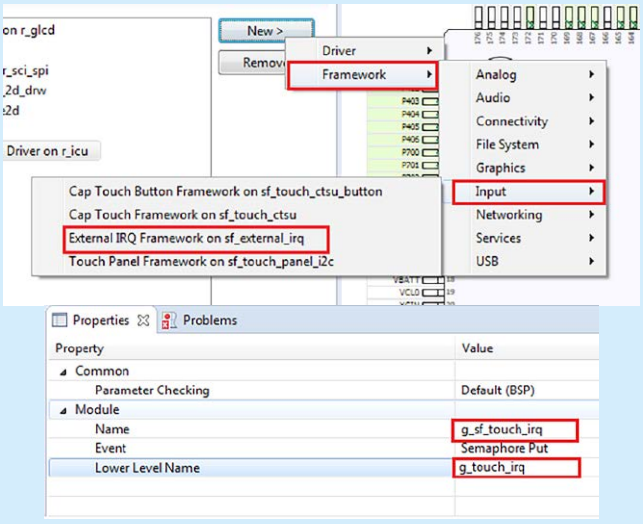
w oknie *Main Thread Modules* dodajemy moduł New → Driver → Connectivity → SCI Common (**rysunek 15**), a następnie New → Driver → SPI Driver on r_rspi. Dodanie tego ostatniego oraz jego konfigurację pokazano na **rysunku 16**. W kolejnych dwóch krokach dodajemy framework i driver D/AVE 2D warstwy framework (**rysunki 17** i **18**).

Jak wspomniano, obsługę interfejsu graficznego zapewnia w bibliotece SSP port GUIX. GUIX zawiera procedury zapewniające wsparcie projektowania interfejsów graficznych za pomocą zewnętrznego narzędzia GUIX Studio – programu dla komputera PC pozwalającego na szybkie projektowanie ekranów interfejsu graficznego z dodawaniem widżetów i przypisywanie im akcji w połączeniu z obsługą ekranu dotykowego. Dodanie funkcji framework GUIX oraz ich konfigurację pokazano na **rysunkach 19** i **20**.

W kolejnych krokach trzeba dodać funkcje do obsługi interfejsu dotykowego, do którego obsługi będzie używane przerwanie zewnętrzne. Interfejs dotykowy jest oparty na układzie scalonym SX8656. Zmiana stanu czujnika dotykowego generuje sygnał



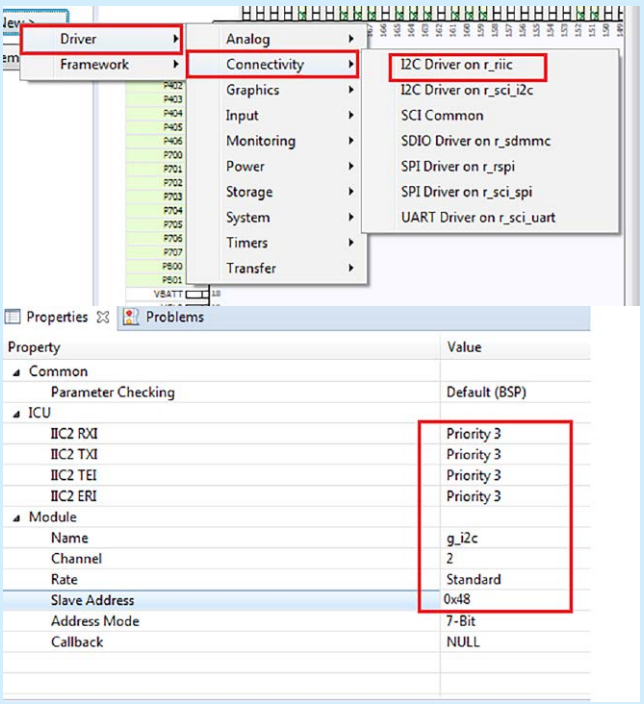
Rysunek 21. Dodanie i konfigurowanie External IRQ Driver on r_icu



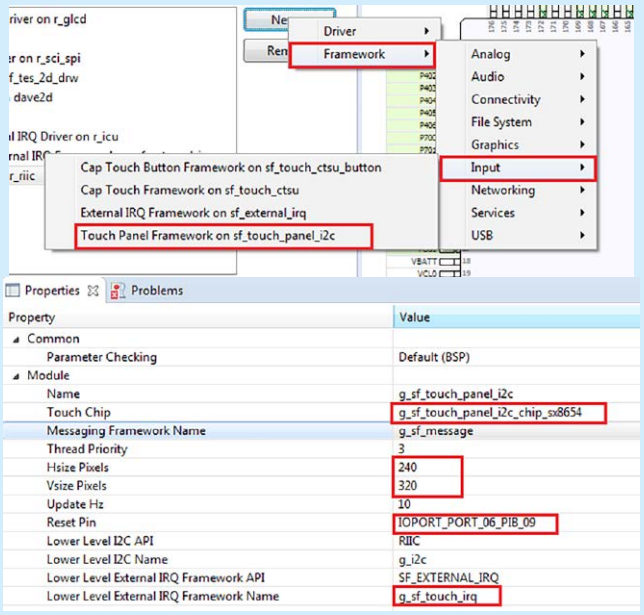
Rysunek 22. Dodanie funkcji frameworka Input Externel IRQ Reamework on sf_external_irq

przerwania IRQ94, a mikrokontroler przez interfejs I²C musi odczytać nowy stan czujnika. Żeby obsłużyć przerwanie zewnętrzne i aby to przerwanie mogło synchronizować działanie wątków RTOS, trzeba dodać funkcje drivera i funkcje framework do obsługi zewnętrznego przerwania IRQ, jak na **rysunkach 21** i **22**.

Następnie dodajemy funkcje warstwy HAL do obsługi interfejsu I²C. We właściwościach drivera podajemy adres slave 0x48 układu kontrolera panelu dotykowego SX8656. Przy zmianie wszystkich pól *name* (nazwa) trzeba bardzo dokładnie wpisać nazwy modułów, bo potem do tych nazw będziemy się odnosili w funkcjach warstwy wyższej. Po dodaniu i konfiguracji funkcji obsługi przerwania zewnętrznego i interfejsu I²C trzeba dodać funkcje framework do obsługi panelu dotykowego. We właściwościach podajemy parametry, które będą potrzebne warstwie BSP: rozdzielczość pionową i poziomą w pikselach oraz pin mikrokontrolera przyłączony do zerowania sterownika panelu.



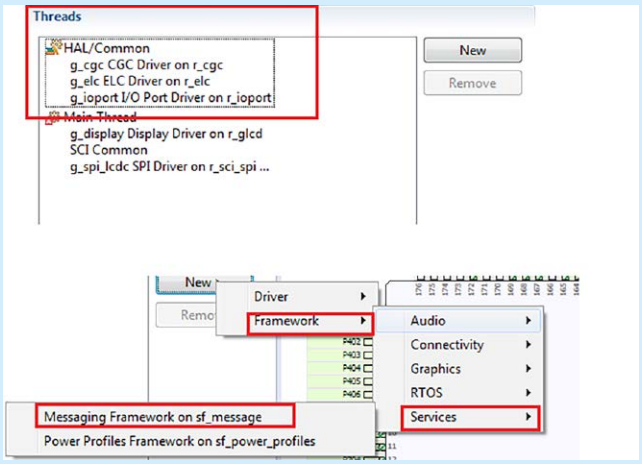
Rysunek 23. Dodanie i konfigurowanie drivera interfejsu I²C warstwy HAL



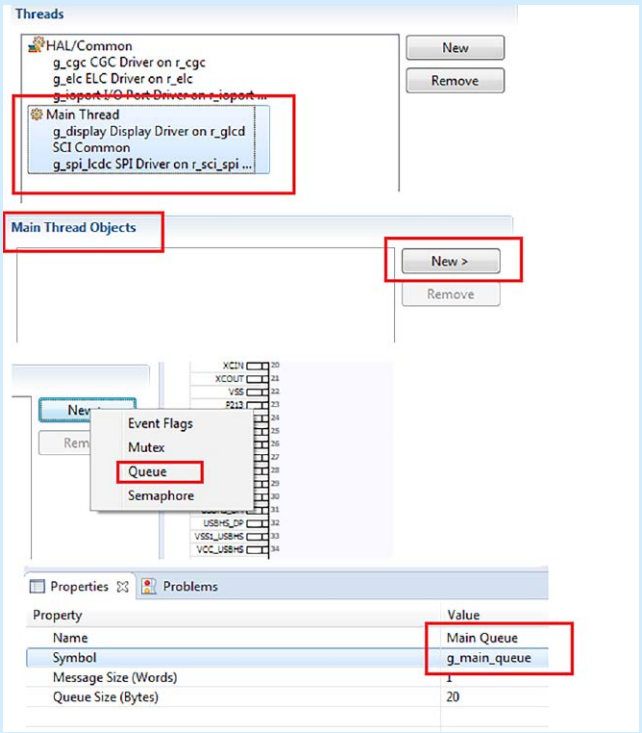
Rysunek 24. Dodanie funkcji frameworka panelu dotykowego

Na tym kończymy dodawanie i konfigurację funkcji drivera i frameworka biblioteki SSP niezbędnych do działania naszej aplikacji i przechodzimy do konfigurowania interfejsów komunikacyjnych i wyprowadzeń mikrokontrolera niezbędnych do poprawnego działania warstwy BSP. Na kolejnych rysunkach są pokazane kroki niezbędne do pracy systemu RTOS (**rysunki 25**, **26** i **27**).

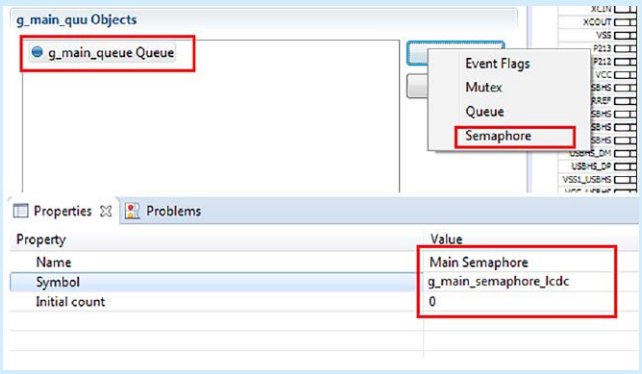
Tomasz Jabłoński, EP



Rysunek 25. Dodanie funkcji frameworka Messaging



Rysunek 26. Dodanie obiektu kolejki



Rysunek 27. Dodanie semafora

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 39483, pass: 5kc7a2ku