Internet of Things (IoT)

Renesas Synergy – Internet Rzeczy w zasięgu ręki (2)

W poprzednim artykule opisaliśmy bazę sprzętową – platformę S7G2 Starter Kit. Teraz opiszemy oprogramowanie oraz przykład mikrokontrolerowego „Hello World" – program migający diodą LED. Jego przygotowanie wymaga użycia środowiska programowego wyposażonego w edytor, kompilator, debugger oraz programy narzędziowe. Na nieskomplikowanym przykładzie pokażemy sposób ich użycia.

Każdy układ mikroprocesorowy jest tak dobry, jak dobre jest jego oprogramowanie. Standardem stało się oferowanie przez producentów mikrokontrolerów modułów ewaluacyjnych. Takim samym standardem jest też zapewnianie bezpłatnego wsparcia. Początkowo były to zintegrowane środowiska IDE, w których można było tworzyć projekt, edytować pliki źródłowe i korzystać z kompilatora asemblera. Teraz jest to zestaw kompletnych narzędzi z bezpłatnym kompilatorem i możliwością zaawansowanego debugowania. Takie bezpłatne narzędzia są często porównywalne użytkowo z bardzo drogimi pakietami płatnymi firm trzecich. Jedyna różnica (jednak często ważna w zastosowaniach profesjonalnych) to brak systemu wsparcia (support). Dla mikrokontrolerów Synergy Renesas przygotował środowisko projektowe **e2studio**. Pobieramy je ze strony Rensasa.

Dodatkowe informacje

Dystrybutorem mikrokontrolerów i zestawów Renesas jest firma **GLYN** mieszczącą się we Wrocławiu przy ul. Krupniczej 13, e-mail: biuro@glyn.pl, www.glyn.pl, telefon: 71-78-28-758).

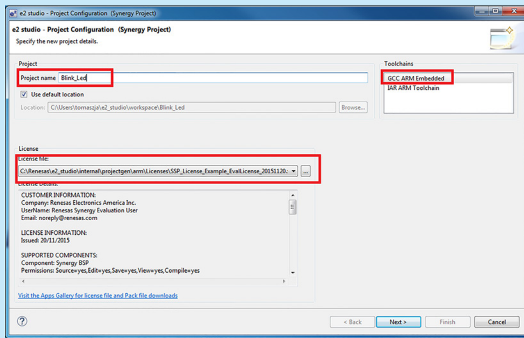
Pakiet jest bezpłatny, ale żeby go pobrać, trzeba się zarejestrować (utworzyć konto).

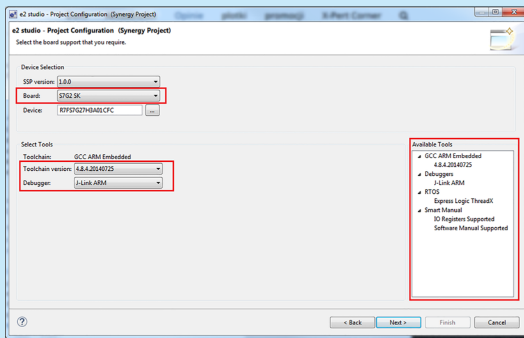
Instalacja przebiega standardowo, jednak trzeba pamiętać by wybrać *Device Families* → *Renesas Synergy*, jak pokazano na **rysunku 10**. Następnie, w sekcji *Select Additional Software*, trzeba zaznaczyć *GCC ARM Embedded 4.8 2014q3* (**rysunek 11**). Oprócz e2studio musimy też pobrać i zainstalować pakiet **Synergy Software Package SSP**. Pierwszą, zalecaną czynnością po zainstalowaniu jest wykonanie aktualizacji, ponieważ wersja instalacyjna może nie zawierać najnowszych wersji wszystkich komponentów. Po uruchomieniu należy wywołać polecenie *Check Updates* wywoływane

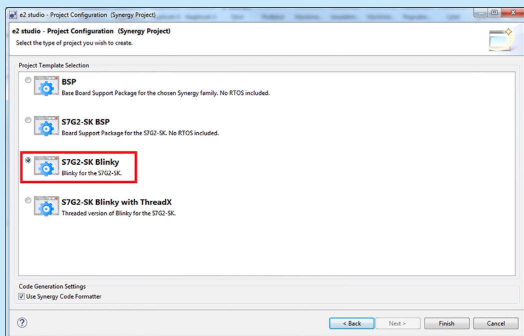
Rysunek 10. Wybór wsparcia dla Synergy w trakcie instalowania e2studio

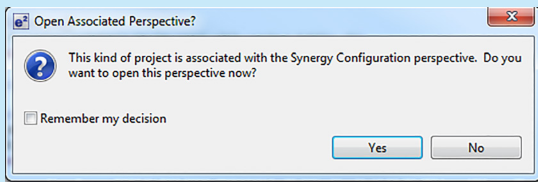
Rysunek 11. Wybór kompilatora GCC ARM

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **86118**, pass: **8655327a**

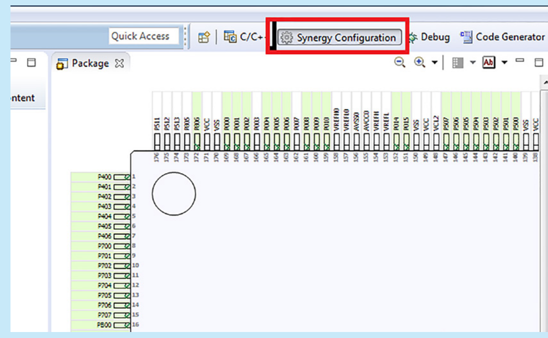

Rysunek 12. Okno konfiguracji projektu

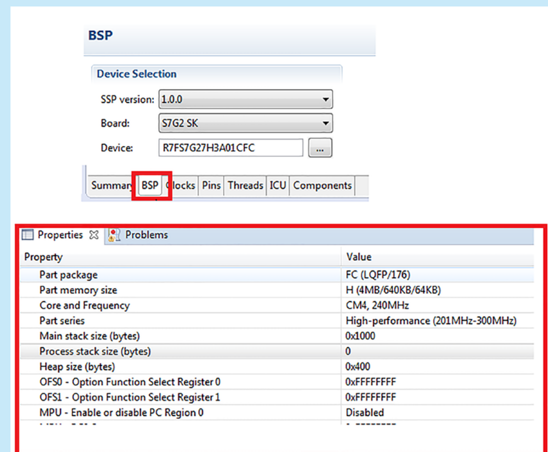

Rysunek 13. Kolejne okno konfiguracji projektu


Rysunek 14. Wybór szablonu projektu

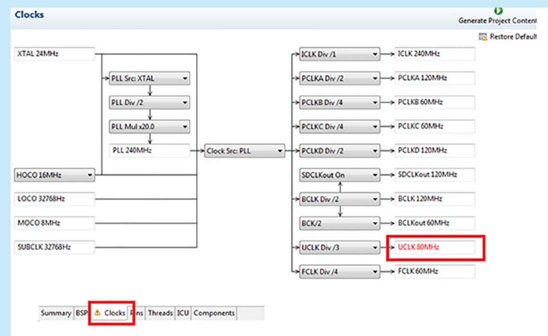


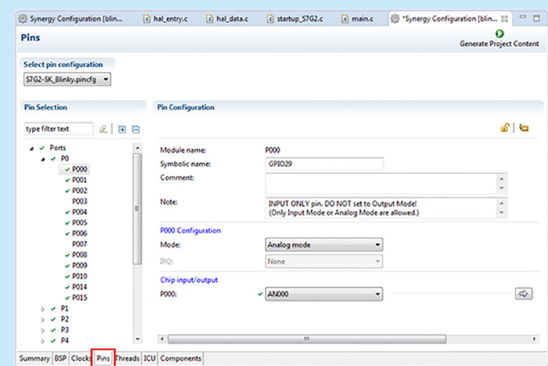
Rysunek 15. Wybór otwarcia perspektywy konfiguracji


Rysunek 16. Otwieranie perspektywy konfiguracji



Rysunek 17. Zakładka ustawień BSP


Rysunek 18. Zakładka Clock – konfigurowanie taktowania mikrokontrolera

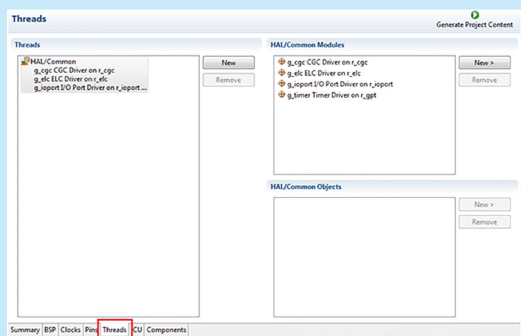

Rysunek 19. Zakładka Pins

z menu *Help*. Wyszukiwanie najnowszych wersji wymaga połączenia komputera z Internetem. Po zainstalowaniu aktualizacji pakiet jest gotowy do użycia.

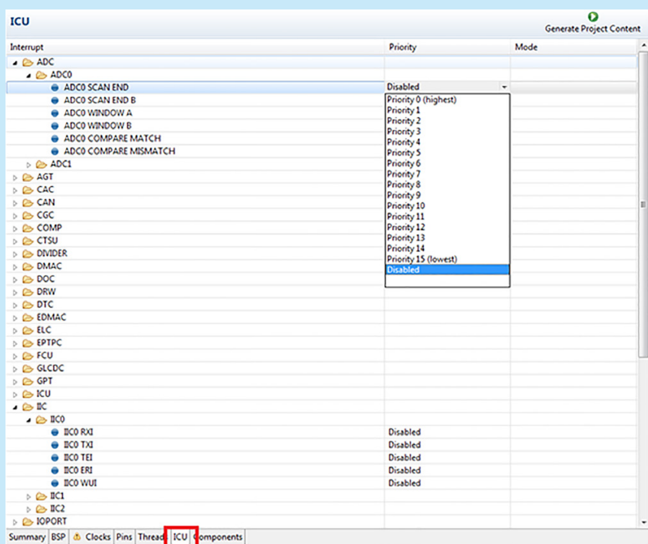
Formalny opis wszystkich okien, menu narzędziowego itp. jest możliwy, ale przynajmniej dla początkującego użytkownika mało przydatny. Najlepiej jest pokazać na prostym przykładzie tworzenie, kompilowanie i uruchomienie prostego przykładu. Bardziej zaawansowane funkcje e2studio (a jest ich sporo) można poznawać stopniowo w miarę potrzeb. Dlatego krok po kroku pokażę sposób utworzenia projektu dla mikrokontrolerów z rodziny Synergy.

Pracę rozpoczynamy od wybrania z menu *File* polecenia *New* → *Synergy Project*. Otwiera się wtedy okno *Project Configuration* pokazane na **rysunku 12**, w którym musimy wpisać nazwę projektu w polu *Project Name* i wybrać folder dla plików projektu w polu *Location*. Po zaznaczeniu opcji *Use default location* projekt zostanie zapisany w lokalizacji domyślnej, przyjętej w momencie instalacji. Przy pierwszym projekcie jesteśmy proszeni o podanie ścieżki dostępu do pliku licencji kompilatora GCC. Jeżeli pakiet kompilatora został prawidłowo zainstalowany, to plik licencji jest zapisany w folderze domyślnym *Renesas\e2_studio\internal\projectgen\arm\Licenses*. Przy

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 86118, pass: 8655327a



Rysunek 20. Zakładka Threads



Rysunek 21. Zakładka ICU

tworzeniu kolejnych projektów nie musimy wskazywać pliku licencji.

Po zaakceptowaniu zmian jest wyświetlane okno konfiguracji projektu pokazane na **rysunku 13**. W tym oknie wybieramy moduł ewaluacyjny. W rozwijanej liście *Board* znajdujemy i zaznaczamy moduł SK-S7G2. Jednocześnie w oknie *Device* zostaje wyświetlony typ wybranego mikrokontrolera. Ponieważ mamy zainstalowany tylko bezpłatny kompilator GCC, to sekcja *Select Tools* pozostaje z ustawieniami domyślnymi. W oknie *Available Tools* są wyświetlane dostępne narzędzia dla naszego projektu: kompilator **GCC ARM**, programator/debugger **J-Link ARM**, **RTOS Express Logic ThreadX**, **IO Register Supported** oraz **Software Manual Support**.

W kolejnym kroku wybieramy szablon projektu **S7G2-SK Blinky** (**rysunek 14**). Po kliknięciu na przycisk *Finish* zostaje wyświetlone okno z pytaniem o otwarcie perspektywy konfiguracyjnej (**rysunek 15**). Klikamy na *Yes* i e2studio tworzy projekt według wybranego szablonu w pespektywie konfiguracyjnej.

Synergy Configuration

Mikrokontroler mający rozbudowane układy peryferyjne, systemy przerwań i taktowania wymaga wstępnej konfiguracji. Konfigurowanie zwykle wykonuje się zapisując rejestry konfiguracyjne. Im bardziej skomplikowany mikrokontroler, tym bardziej żmudne jest jego konfigurowanie. Aby ułatwić pracę programistom wiele programów narzędziowych IDE

Listing 1. Funkcja main()

```
extern void hal_entry(void);
int main(void)
{
    hal_entry ();
    return 0;
}
```

Listing 2. Funkcja migania diodami LED

```
#include „hal_data.h"
/*******************************************************
 * @brief  Blinky example application
 *
 * Blinks all leds at a rate of 1 second using the
 * software delay function provided by the BSP.
 * Only references two other modules including the BSP, IOPORT.
 *
*******************************************************/
void hal_entry(void) {
/* Define the units to be used with the software delay function */
    const bsp_delay_units_t bsp_delay_units = BSP_DELAY_UNITS_MILLISECONDS;
/* Set the blink frequency (must be <= bsp_delay_units */
    const uint32_t freq_in_hz = 2;
/* Calculate the delay in terms of bsp_delay_units */
    const uint32_t delay = bsp_delay_units/freq_in_hz;
/* LED type structure */
    bsp_leds_t leds;
/* LED state variable */
    ioport_level_t level = IOPORT_LEVEL_HIGH;
    /* Get LED information for this board */
    R_BSP_LedsGet(&leds);
    /* If this board has no LEDs then trap here */
    if (0 == leds.led_count)
    {
        while(1);   // There are no LEDs on this board
    }
    while(1)
    {
        /* Determine the next state of the LEDs */
        if(IOPORT_LEVEL_LOW == level)
        {
            level = IOPORT_LEVEL_HIGH;
        }
        else
        {
            level = IOPORT_LEVEL_LOW;
        }
        /* Update all board LEDs */
        for(uint32_t i = 0; i < leds.led_count; i++)
        {
            g_ioport.p_api->pinWrite(leds.p_leds[i], level);
        }
        /* Delay */
        R_BSP_SoftwareDelay(delay, bsp_delay_units);
    }
}
```

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 86118, pass: 865327a

ma wbudowywane konfiguratory, które znacznie tę pracę ułatwiają. Konfigurator wbudowany w e2studio nazywa się **Synergy Configurator**, a jego ustawienia są zapisywane w pliku configuration.xml. Otwarcie tego pliku powoduje otwarcie perspektywy konfiguracyjnej z konfiguracją projektu.

Rysunek 22. Zakładka Components

Rysunek 23. Struktura plików projektu

Rysunek 24. Ustawienie w trybie Debug

Pespektywę konfiguracyjna otwieramy po kliknięciu na przycisk *Synergy Configuration*, jak pokazano na **rysunku 16**. Konfiguracje są podzielone funkcjonalnie na etapy wybierane za pomocą zakładek:

- **BSP**. Wybiera się tu wersję biblioteki BSP, moduł ewaluacyjny i mikrokontroler. Te ustawienia zostały wskazane w trakcie generowania projektu Synergy, ale tu można je zmienić. Z zakładką jest skojarzone okno *Properties* (**rysunek 17**), w którym są wyświetlane ustawienia dotyczące wybranego mikrokontrolera. Te ustawienia można zmieniać z poziomu okna *Properties*.
- **Clock**. Przeznaczona do skonfigurowania systemu taktowania mikrokontrolera. Zakładka jest przedstawiona w formie graficznej, co znacznie ułatwia konfigurowanie taktowania. Jeżeli któreś z ustawień nie jest możliwe do uzyskania, na przykład – taktowanie modułu USB częstotliwością 48 MHz, to ustawienie jest wyświetlane na czerwono, jak na **rysunku 18**.
- **Pins**. Przeznaczona do graficznego konfigurowania funkcji wyprowadzeń mikrokontrolera. Wyprowadzenia mogą być konfigurowane w dwóch grupach – jako linie portów lub jako wyprowadzenia układów peryferyjnych (**rysunek 19**).
- **Threads**. Przeznaczona do dodawania i konfigurowania komponentów, które chcielibyśmy użyć w projekcie. Ponieważ w naszym projekcie nie używamy RTOS poprzestaniemy na domyślnym HAL/Common (**rysunek 20**).
- **ICU** (Interrupt Controller Unit). Pokazuje wszystkie źródła przerwań od układów peryferyjnych i umożliwia skonfigurowanie przerwań (**rysunek 21**).
- **Components**. Wyświetla komponenty biblioteki SSP oraz dołączone do projektu (**rysunek 22**). Komponenty mogą być dołączane za pomocą zakładki *Threads*.

Wszystkie generowane konfiguracje są umieszczane w plikach źródłowych projektu po kliknięciu na *Generate Project Content*. Te pliki są umieszczane w folderach o nazwach: **synergy_gen**, **synergy** i **synergy_cfg**. Nie powinno się plików konfiguracyjnych modyfikować ręcznie. Jeżeli trzeba coś zmienić, to należy to zrobić za pomocą programu *Synergy Configurator*. W przeciwnym przypadku ręcznie naniesione zmiany zostaną nadpisane przy kolejnym uruchomieniu konfiguratora.

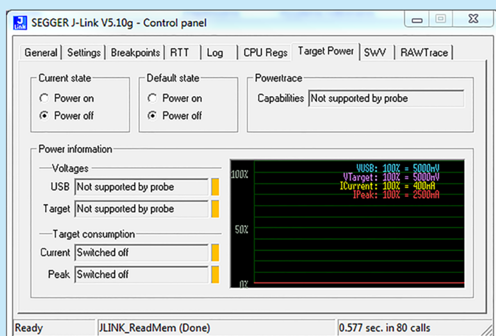
W katalogu **synergy_cfg** są umieszczone pliki konfigurujące komponenty projektu *Synergy*. Te konfiguracje bazują na ustawieniach zapisanych w perspektywie *Synergy Configuration*. Katalog **synergy** zawiera pliki źródłowe wydzielone z biblioteki SSP przeznaczone dla komponentów aktualnie używanych w projekcie. Konfiguracja tych komponentów jest umieszczona również w katalogu **synergy_cfg**. Katalog **synergy_gen** zawiera konfiguracje wcieleń komponentów, które użytkownik dodał do projektu.

Jak pokazano na **rysunku 23**, tylko plik źródłowy *hal_entry.c* jest przeznaczony do edytowania przez użytkownika. Nawet plik *main.c* jest generowany przez konfigurator i nie powinien być edytowany przez użytkownika.

Wróćmy do naszego projektu, którego zadaniem jest miganie diodą LED. Konfigurator projektu wygenerował między innymi plik źródłowy **hal_entry.c** przeznaczony do modyfikacji przez użytkownika, umieszczony w katalogu **src** oraz plik **main.c** umieszczony w katalogu **src/synergy_gen**. Funkcję główną *main()* pokazano na **listingu 1**.

Funkcja *hal_entry.c* jest przeznaczona do edycji i jest raczej nieskomplikowana. Odczytuje poziom występujący

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 86118, pass: 8655327a



Rysunek 25. Okno sterujące sprzętowego debuggera Segger J-Link

na liniach portów sterujących trzema diodami LED zamontowanymi na płytce SK-S7G2, zmienia go na przeciwny i odlicza opóźnienie ok 1 sekundy. Funkcja biblioteczna *R_BSP_SoftwareDelay* odlicza opóźnienie programowo – nie używa do tego celu liczników sprzętowych ani układu przerwań.

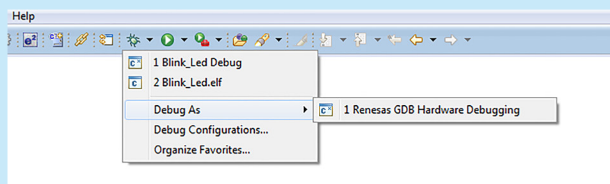
Jak wiemy, w projekcie zostały umieszczone pliki konfiguracyjne wygenerowane przez konfigurator Synergy. Konfigurowanie nie jest jawnie uruchamiane w programie głównym.

Listing 3. Funkcja Reset_Handler

```
/*******************************************************
* Function Name: Reset_Handler
* Description: MCU starts executing here out of reset.
* Main stack pointer is setup already.
* Arguments    : none
* Return Value : none
********************************************************/
void Reset_Handler (void)
{
    /* Initialize system using BSP. */
    SystemInit();

    /* Call user application. */
    main();

    while (1)
    {
        /* Infinite Loop. */
    }
}
```



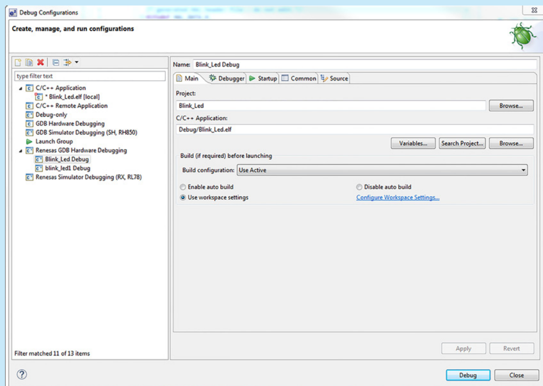
Rysunek 26. Uruchamianie debugowania

Listing 4. Funkcja inicjalizacji mikrokontrolera

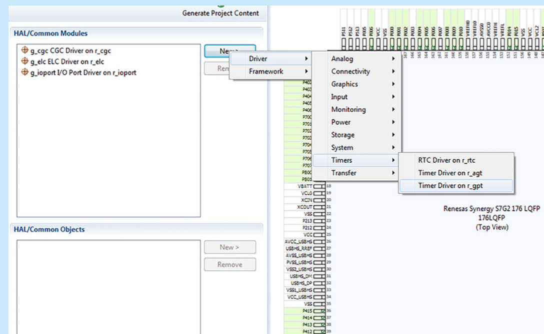
```
/*******************************************************
* Function Name: SystemInit
* Description  : Setup MCU.
* Arguments    : none
* Return Value : none
********************************************************/
void SystemInit (void)
{
#if ( defined ( __GNUC__ ) && defined (__VFP_FP__) && !defined (__SOFTFP__) ) || \
    ( defined ( __ICCARM__ ) && defined ( __ARMVFP__ ) && (__FPU_PRESENT == 1) )
    /* Enable the Cortex-M4 FPU only when -mfloat-abi=hard.
       Code taken from Section 7.1, Cortex-M4 TRM (DDI0439C) */
    /* Set bits 20-23 to enable CP10 and CP11 coprocessor */
    SCB->CPACR |= (0xF << 20);
#endif

    /* Call Pre C runtime initialization hook. */
    R_BSP_WarmStart(BSP_WARM_START_PRE_C);
    /* Initialize grouped interrupts. */
    bsp_group_interrupt_open();
    /* Configure system clocks using CGC module. */
    bsp_clock_init();
    /* Initialize register protection. */
    bsp_register_protect_open();
    /* Handle VBTICTLR register. */
    bsp_vbatt_init(&g_bsp_pin_cfg);
    /* Initialize pins. */
    g_ioport_on_ioport.init(&g_bsp_pin_cfg);
    /* Initialize C runtime environment. */
    /* Zero out BSS */
#if defined(__GNUC__)
    bsp_section_zero((uint8_t *)&__bss_start__, ((uint32_t)&__bss_end__ - (uint32_t)&__bss_start__));
#elif defined(__ICCARM__)
    bsp_section_zero((uint8_t *)__section_begin(„.bss”), (uint32_t)__section_size(„.bss”));
#endif
    /* Copy initialized RAM data from ROM to RAM. */
#if defined(__GNUC__)
    bsp_section_copy((uint8_t *)&__etext,
                     (uint8_t *)&__data_start__,
                     ((uint32_t)&__data_end__ - (uint32_t)&__data_start__));
#elif defined(__ICCARM__)
    bsp_section_copy((uint8_t *)__section_begin(„.data_init”),
                     (uint8_t *)__section_begin(„.data”),
                     (uint32_t)__section_size(„.data”));
    /* Copy functions to be executed from RAM. */
    #pragma section=".code_in_ram"
    #pragma section=".code_in_ram_init"
    bsp_section_copy((uint8_t *)__section_begin(„.code_in_ram_init”),
                     (uint8_t *)__section_begin(„.code_in_ram”),
                     (uint32_t)__section_size(„.code_in_ram”));
#endif
    /* Initialize SystemCoreClock variable. */
    SystemCoreClock = bsp_cpu_clock_get();
    /* Call Post C runtime initialization hook. */
    R_BSP_WarmStart(BSP_WARM_START_POST_C);
    /* Initialize the Hardware locks to 'Unlocked' */
    bsp_init_hardware_locks();
    /* Initialize ELC events that will be used to trigger NVIC interrupts. */
    bsp_irq_cfg();
    /* Initialize ELC. */
    g_elc_on_elc.init(&g_elc_cfg);
    /* Call any BSP specific code. No arguments are needed so NULL is sent. */
    bsp_init(NULL);
}
```

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **86118**, pass: **8655327a**



Rysunek 27. Okno debugowania konfiguracji



Rysunek 28. Dodanie drivera GPT

Żeby zobaczyć jak przebiega konfiguracja wykorzystamy możliwość sprzętowego debugowania programu. Projekt musi być ustawiany w trybie **Debug**. Robimy to klikając obok ikonki młotka w pasku narzędzi (**rysunek 24**). Następnie kompilujemy projekt. Po bezbłędnym skompilowaniu możemy przejść do debugowania. Łączymy moduł kablem USB ze złączem microUSB (J19 – DEBUG_USB) z portem USB komputera PC. Po automatycznym zainstalowaniu się sterownika J-Link można przejść do sprzętowego debugowania programu. Jeśli debugger J-link jest przyłączony

Listing 5. Konfigurowanie taktowania mikrokontrolera

```
/*****************************************************
* @brief      Sets up system clocks.
/*****************************************************
void bsp_clock_init (void)
{
    g_cgc_on_cgc.init();
    /** MOCO is default clock out of reset. Enable new clock if chosen. */
    cgc_clock_t clock;
    if (BSP_CFG_CLOCK_SOURCE != CGC_CLOCK_PLL)
    {
        clock = BSP_CFG_CLOCK_SOURCE;
        g_cgc_on_cgc.clockStart(clock, NULL);
    }
    else
    {
        /** Need to start PLL source clock and let it stabilize before starting PLL */
        clock = BSP_CFG_PLL_SOURCE;
        g_cgc_on_cgc.clockStart(clock, NULL);
        cgc_clock_cfg_t pll_cfg;
        /** Set PLL Divider. */
        pll_cfg.divider = BSP_CFG_PLL_DIV;
        /** Set PLL Multiplier. */
        pll_cfg.multiplier = BSP_CFG_PLL_MUL;
        /** Set PLL Source clock. */
        pll_cfg.source_clock = clock;
        while (SSP_ERR_STABILIZED != g_cgc_on_cgc.clockCheck(clock))
        {
            /** Wait for PLL clock source to stabilize */
        }
        g_cgc_on_cgc.clockStart(CGC_CLOCK_PLL, &pll_cfg);
        clock = CGC_CLOCK_PLL;
    }
    R_ROMC->ROMCE_b.ROMCEN = 1;          /** Enable ROM cache */
    /** MOCO, LOCO, and subclock do not have stabilization flags that can be checked. */
    if ((CGC_CLOCK_MOCO != clock) && (CGC_CLOCK_LOCO != clock) && (CGC_CLOCK_SUBCLOCK != clock))
    {
        while (SSP_ERR_STABILIZED != g_cgc_on_cgc.clockCheck(clock))
        {
            /** Wait for clock source to stabilize */
        }
    }
    cgc_system_clock_cfg_t sys_cfg;
    sys_cfg.iclk_div  = BSP_CFG_ICK_DIV;
    sys_cfg.pclka_div = BSP_CFG_PCKA_DIV;
    sys_cfg.pclkb_div = BSP_CFG_PCKB_DIV;
    sys_cfg.pclkc_div = BSP_CFG_PCKC_DIV;
    sys_cfg.pclkd_div = BSP_CFG_PCKD_DIV;
    sys_cfg.fclk_div  = BSP_CFG_FCK_DIV;
    sys_cfg.bclk_div  = BSP_CFG_BCK_DIV;
    /** Set which clock to use for system clock and divisors for all system clocks. */
    g_cgc_on_cgc.systemClockSet(clock, &sys_cfg);
    /** Set USB clock divisor. */
    g_cgc_on_cgc.usbClockCfg(BSP_CFG_UCK_DIV);
    /** Configure BCLK */
#if   BSP_CFG_BCLK_OUTPUT == 1
    g_cgc_on_cgc.busClockOutCfg(CGC_BCLOCKOUT_DIV_1);
    g_cgc_on_cgc.busClockOutEnable();
#elif BSP_CFG_BCLK_OUTPUT == 2
    g_cgc_on_cgc.busClockOutCfg(CGC_BCLOCKOUT_DIV_2);
    g_cgc_on_cgc.busClockOutEnable();
#else
    g_cgc_on_cgc.busClockOutDisable();
#endif
    /** Configure SDRAM Clock */
#if   BSP_CFG_SDCLK_OUTPUT == 0
    g_cgc_on_cgc.sdramClockOutDisable();
#else
    g_cgc_on_cgc.sdramClockOutEnable();
#endif
}
```

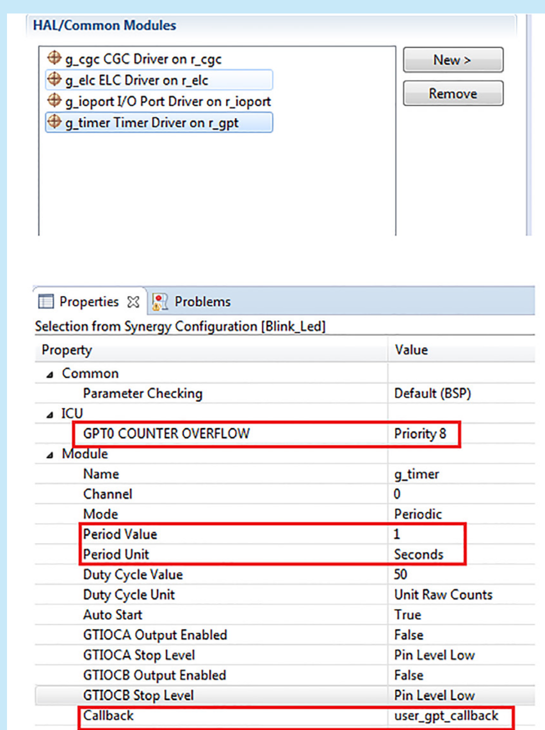
Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 86118, pass: 8655327a

```
Listing 6. Plik nagłówkowy bsp_clock_cfg.h
/* generated configuration header file - do not edit */
#ifndef BSP_CLOCK_CFG_H_
#define BSP_CLOCK_CFG_H_
#define BSP_CFG_XTAL_HZ (24000000) /* XTAL 24000000Hz */
#define BSP_CFG_PLL_SOURCE (CGC_CLOCK_MAIN_OSC) /* PLL Src: XTAL */
#define BSP_CFG_HOCO_FREQUENCY (0) /* HOCO 16MHz */
#define BSP_CFG_PLL_DIV (CGC_PLL_DIV_2) /* PLL Div /2 */
#define BSP_CFG_PLL_MUL (20.0) /* PLL Mul x20.0 */
#define BSP_CFG_CLOCK_SOURCE (CGC_CLOCK_PLL) /* Clock Src: PLL */
#define BSP_CFG_ICK_DIV (CGC_SYS_CLOCK_DIV_1) /* ICLK Div /1 */
#define BSP_CFG_PCKA_DIV (CGC_SYS_CLOCK_DIV_2) /* PCLKA Div /2 */
#define BSP_CFG_PCKB_DIV (CGC_SYS_CLOCK_DIV_4) /* PCLKB Div /4 */
#define BSP_CFG_PCKC_DIV (CGC_SYS_CLOCK_DIV_4) /* PCLKC Div /4 */
#define BSP_CFG_PCKD_DIV (CGC_SYS_CLOCK_DIV_2) /* PCLKD Div /2 */
#define BSP_CFG_SDCLK_OUTPUT (1) /* SDCLKout On */
#define BSP_CFG_BCK_DIV (CGC_SYS_CLOCK_DIV_2) /* BCLK Div /2 */
#define BSP_CFG_BCLK_OUTPUT (2) /* BCK/2 */
#define BSP_CFG_UCK_DIV (CGC_USB_CLOCK_DIV_5) /* UCLK Div /5 */
#define BSP_CFG_FCK_DIV (CGC_SYS_CLOCK_DIV_4) /* FCLK Div /4 */
#endif /* BSP_CLOCK_CFG_H_ */
```

do komputera, to automatycznie jest wyświetlane okno *Segger J-Link Control Panel* (**rysunek 25**). Debugowanie jest uruchamiane po kliknięciu na ikonkę przypominającą owada i wybraniu *Debug As → Renesas GDB Hardware Debugging*

```
Listing 7. Zmodyfikowany fragment pliku hal_data.h
extern const timer_instance_t g_timer;
#if TIMER_ON_GPT_CALLBACK_USED_g_timer
void user_gpt_callback(timer_callback_args_t * p_args);
#endif
```

```
Listing 8. zmodyfikowany fragment pliku hal_data.c
#pragma weak user_gpt_callback     user_gpt_callback_internal
static timer_ctrl_t g_timer_ctrl;
static const timer_cfg_t g_timer_cfg =
{ .mode = TIMER_MODE_PERIODIC, .period = 1, .unit = TIMER_UNIT_PERIOD_SEC, .duty_cycle = 50, .duty_cycle_unit
=
          TIMER_PWM_UNIT_RAW_COUNTS,
  .channel = 0, .autostart = true, .p_callback = user_gpt_callback, .p_context = &g_timer, .p_extend = &g_
timer_extend };
/* Instance structure to use this module. */
const timer_instance_t g_timer =
{ .p_ctrl = &g_timer_ctrl, .p_cfg = &g_timer_cfg, .p_api = &g_timer_on_gpt };
#if TIMER_ON_GPT_CALLBACK_USED_g_timer
/******************************************************
* @brief   This is a weak example callback function.
* It should be overridden by defining a user callback
* function with the prototype below.
* - void user_gpt_callback(timer_callback_args_t * p_args)
* @param[in]  p_args  Callback arguments used to identify
* what caused the callback.
******************************************************
void user_gpt_callback_internal(timer_callback_args_t * p_args);
void user_gpt_callback_internal(timer_callback_args_t * p_args)
{
    /** Do nothing. */
    SSP_PARAMETER_NOT_USED(p_args);
}
```



Rysunek 29. Właściwości drivera GPT

```
Listing 9. Procedura obsługi przerwania
volatile bool g_timer_flag;
void user_gpt_callback(timer_callback_args_t * p_args)
{
    g_timer_flag = true;
}
```

(**rysunek 26**). Debbuger programu e2studio działa podobnie jak w środowiskach IDE innych producentów. Do sterowania debugowaniem używa się następujących ikonek:

- uruchom program,
- zatrzymaj program,
- zakończ debugowanie,
- odłącz proces debugowania,
- wykonanie jednego kroku programu z wejściem do funkcji,
- wykonanie całej funkcji,
- krok powrotu,
- debugowanie instrukcji assemblera,
- usuń wszystkie punkty zatrzymania (breakpoints),
- zapisz program do pamięci,
- zeruj debugger,
- zeruj mikrokontroler (działanie programu),
- odśwież.

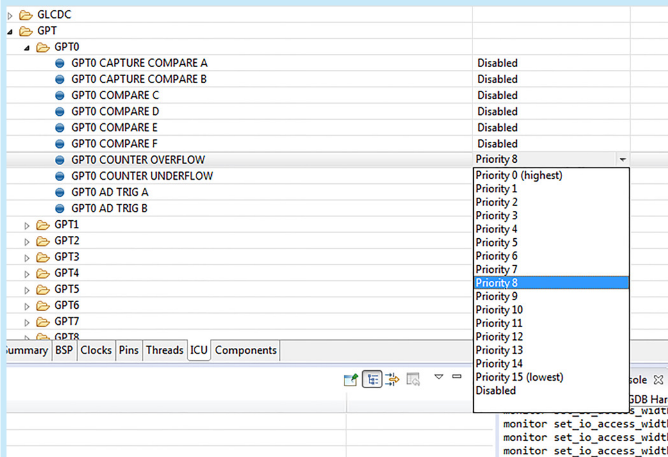
Po uruchomieniu debugowania automatycznie są wyświetlane okna: *Debug*, *Project Explorer*, okno

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 86118, pass: 8655327a

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 86118, pass: 8655327a

Listing 10. Miganie diodami ze zmodyfikowanym odliczaniem opóźnień

```
ssp_err_t err;
/* Define the units to be used with the software delay function */
    const bsp_delay_units_t bsp_delay_units = BSP_DELAY_UNITS_MILLISECONDS;
/* Set the blink frequency (must be <= bsp_delay_units */
    const uint32_t freq_in_hz = 2;
/* Calculate the delay in terms of bsp_delay_units */
    const uint32_t delay = bsp_delay_units/freq_in_hz;
/* LED type structure */
    bsp_leds_t leds;
/* LED state variable */
    ioport_level_t level = IOPORT_LEVEL_HIGH;
    /* Get LED information for this board */
    R_BSP_LedsGet(&leds);
    err = g_timer.p_api->open(g_timer.p_ctrl, g_timer.p_cfg);
    if (SSP_SUCCESS != err)
    {
    while(1);
    }
    /* If this board has no LEDs then trap here */
    if (0 == leds.led_count)
    {
        while(1);   // There are no LEDs on this board
    }
    while(1)
    {
        /* Determine the next state of the LEDs */
        if(IOPORT_LEVEL_LOW == level)
        {
            level = IOPORT_LEVEL_HIGH;
        }
        else
        {
            level = IOPORT_LEVEL_LOW;
        }
        /* Update all board LEDs */
        for(uint32_t i = 0; i < leds.led_count; i++)
        {
            g_ioport.p_api->pinWrite(leds.p_leds[i], level);
        }
        /* Delay */
        while (false == g_timer_flag);
        g_timer_flag = false;
        //R_BSP_SoftwareDelay(delay, bsp_delay_units);
    }
```



Rysunek 30. Konfigurowanie przerwań od drivera GPT0

z debugowanym kodem i *Disassembly*. W oknie z debugowanym kodem będziemy oglądać wykonywanie programu napisanego w C, a w oknie *Dissasembly* kod w asemblerze. Teraz możemy zobaczyć jak się wykonuje nasz program z **listingu 2**. Wskaźnik rozkazów nie zatrzymuje się na wywołaniu pierwszej procedury z funkcji *main()*, ale na pierwszej procedurze funkcji *Reset_Handler()* , którą pokazano na **listingu 3**. Jak się łatwo domyślić, jest to funkcja wywoływana po zerowaniu mikrokontrolera i to dopiero ona wywołuje funkcję główną *main()*. Jednak zanim to nastąpi, wywoływana jest funkcja *System Init()* umieszczona w pliku startup_S7G2.c (**listingu 4**).

Zgodnie z przewidywaniami inicjalizacja jest dość rozbudowana i zawiera procedury ustawiające między innymi taktowanie, system przerwań, protekcję zapisu rejestrów, inicjowanie linii GPIO itp. Każdą z funkcji inicjalizujących można w debugerze wykonywać krokowo i posiłkując się dokumentacją mikrokontrolera patrzeć jak postępuje konfigurowanie mikrokontrolera. Możemy jako przykład zobaczyć jak wygląda ustawianie taktowania mikrokontrolera wykonywane przez procedurę *bsp_clock_init* (**listing 5**). Ustawienia taktowania są zapisane w pliku *bsp_clock_cfg.h* (**listing 6**). Istnieje też możliwość debugowania konfiguracji projektu. Zamiast *Debug As* klikamy na *Debug Configurations* i w nim na zakładkę *Debug* (**rysunek 27**). Po kliknięciu na przycisk *Debug* jest wyświetlane okno z plikiem *startup_S7G2.c* (**rysunku 28**).

Na koniec pokażę jak dodać nowy komponent do konfiguracji projektu. Będzie to sprzętowy timer GPT odliczający opóźnienia. Ma on zastąpić programowego odliczania opóźnień w naszym projekcie. Jak wiemy dodawanie komponentów może by wykonywane za pomocą zakładki *Threads*. Najpierw konfigurujemy przerwania od GPT0 zgłaszane przy przepełnieniu licznika (**rysunek 29**). Potem we właściwościach licznika (zakładka Threads) wpisujemy: Period value=1, Period unit = seconds, Callback = user_gpt_callback (**rysunek 30**). Tak skonfigurowany Timer przepełnia się i zgłasza przerwanie co 1 sekundę. Teraz na podstawie nowej konfiguracji *Synergy Configurator* wygeneruje nowe pliki konfiguracyjne po kliknięciu na *Generate Project Content*. W pliku *hal_data.h* konfigurator dopisał definicje pokazane na **listingu 7**. A w pliku *hal_data.c* konfigurator dopisał definicję konfiguracji licznika (**listing 8**). Teraz pozostaje napisać tylko procedurę *user_gpr_callback* wywoływaną przy każdym zgłoszeniu przerwania od licznika GPT. Tę procedurę możemy umieścić tylko w pliku *hal_entry.c*, jak pokazano na **listingu 9**. Zmodyfikowana pętla cyklicznie gasząca i zapalająca diody LED na module ewaluacyjnym pokazano na **listingu 10**.

Tomasz Jabłoński, EP