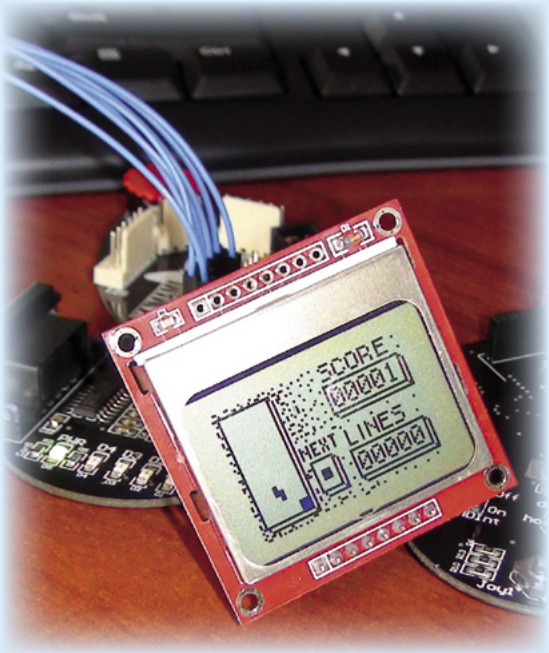
Tetris na STM32Butterfly (1)

W cyklu artykułów zaprezentujemy proces tworzenia klonu gry Tetris na platformę z mikrokontrolerem 32-bitowym z rodziny STM32 firmy STMicroelectronics. W tym artykule opisujemy sposób przygotowania środowiska programistycznego oraz szablonu projektu, opis mechanizmu gry i jego realizację w języku C.

Z pewnością większość czytelników grała w jeden z klonów gry Tetris. Mimo, iż ta gra z Rosji rodem ma już 27 lat to jest w niej to coś, co powoduje, że każdy kto ma z nią styczność zatraca się w niej choćby na kwadrans. Czy łatwo jest napisać jej kolejny klon? Sprawdźmy…



Instalacja środowiska programistycznego

Do wykonania gry użyłem „motyla" pierwszej generacji, wyświetlacza z Nokii 3310 oraz programatora ZL30PRG od Kamami. Takie akurat miałem pod ręką, ale nic nie stoi na przeszkodzie, aby użyć nowszych wersji tego sprzętu, tj. „motyla" 2 i ZL30PRGv2. Artykuł napisałem w taki sposób, aby każdy mógł dostosować program do posiadanego przez siebie sprzętu.

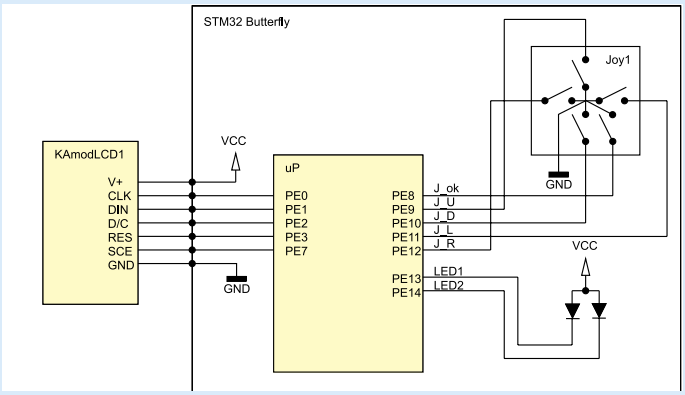
Samo przygotowanie „motyla" i jego otoczenia do pracy jest banalne. Polegać będzie na dołączeniu wyświetlacza do wyprowadzeń portu PE za pomocą tasiemki z dwoma wtykami IDC10 (**rysunek 1)**. Należy zwrócić uwagę, aby piny zasilania wyświetlacza były prawidłowo podłączone do pinów zasilania na „motylu". Ponadto wykorzystamy joystick znajdujący się na płytce motyla. Nie trzeba go dodatkowo konfigurować lub podłączać, od razu jest gotowy do użycia. Pozostaje nam wsadzenie płytki programatora do złącza oznaczonego JTAG.

Teraz pora na konfigurację środowiska programistycznego. Wybrałem TrueSTUDIO for STMicroelectronics STM32 firmy Atollic w wersji Lite, bo jest darmowe i łatwe do zainstalowania. Ponadto, po zainstalowaniu nie będzie wymagało żmudnej konfiguracji.

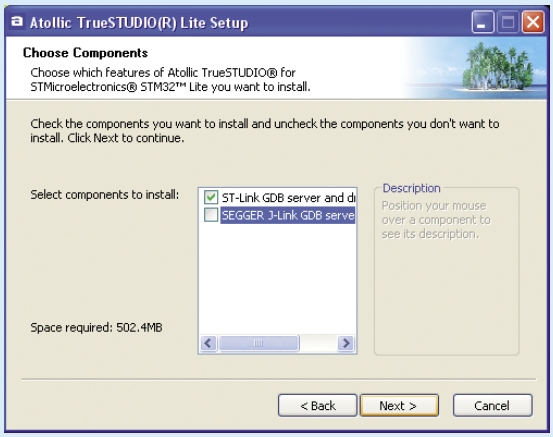
Na stronie internetowej firmy *Atollic* i wybieramy **dział** *Download*, a tam odnajdujemy interesującą nas wersję programu: *http://www.atollic.com/index.php/download/downloadstm32*. Zaznaczamy pole *I accept the License agreement*, po czym klikamy przycisk *Free Download*. Po ściągnięciu pliku instalacyjnego uruchamiamy go. W czasie instalacji zostaniemy poproszenie o zaakceptowanie warunków licencji. Na trzeciej stronie instalatora mamy możliwość wyboru, jakie sterowniki programatorów zostaną zainstalowane. Ja wybrałem tylko ST-Link GDB Server (**rysunek 2**), który współpracuje z programatorem ZL30PRG.

Kolejna strona pozwala określić lokalizację plików programu. W momencie jak kolejny raz naciśniemy przycisk NEXT w przeglądarce internetowej powinna otworzyć się strona z formularzem rejestracyjnym. Rejestracja jest całkowicie darmowa, wymaga podania swoich danych oraz adresu email. W pole Computer ID kopiujemy zawartość okna instalatora o tej samej nazwie (**rysunek 3**).

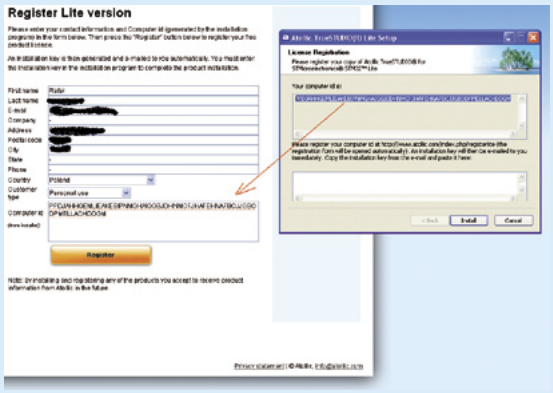
Teraz należy wysłać formularz rejestracyjny naciskając przycisk Register. W ciągu kilku minut powinna dotrzeć do nas wiadomość z kluczem, który kopiujemy w puste pole okna instalatora. Jeżeli wiadomość nie



Rysunek 1. Sposób dołączenia wyświetlacza do płytki STM32Butterfly



Rysunek 2. Wybranie ST-Link GDB Server



Rysunek 3. Wpisanie identyfikatora komputera

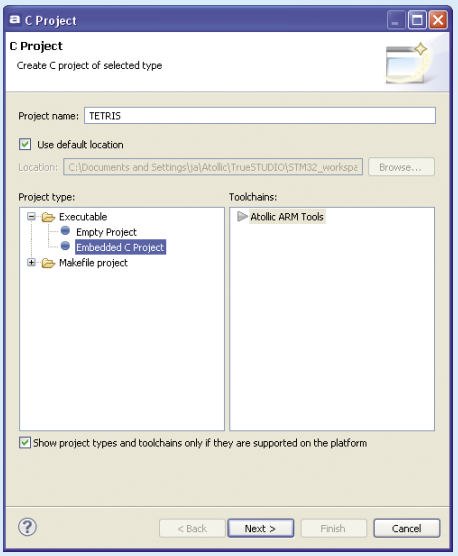
Rysunek 4. Wiadomość z kluczem aktywacyjnym

dotrze do nas warto sprawdzić zakładkę SPAM naszej poczty. W moim przypadku skrzynka założona na znanym portalu właśnie tak sklasyfikowała wiadomość (**rysunek 4**).

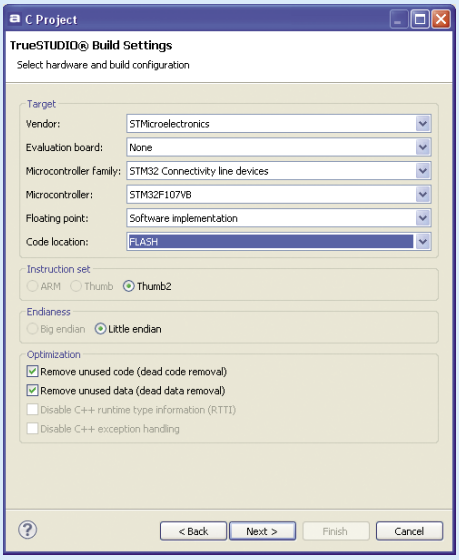
Po skopiowaniu otrzymanego klucza klikamy przycisk Install. Po skopiowaniu plików przez instalator pozostanie nam jeszcze kilka kliknięć przycisku *Next* na kartach z informacjami o programie by w końcu zakończyć proces instalacji.

Tworzymy projekt gry

Jeżeli instalacja przebiegła pomyślnie to po uruchomieniu programu zostaniemy poproszenie o potwierdzenie położenia katalogu roboczego. Nasze środowisko od razu jest gotowe, aby stworzyć w nim nasz projekt. Aby tego dokonać klikamy File->New->C Projekt. W okienku, które się pojawi wpisujemy nazwę projektu, ja wpisałem TETRIS oraz wybieramy jego typ: Embedded C Project. Klikamy *Next* (**rysunek 5**).



Rysunek 5. Wybór projektu w języku C



Rysunek 6. Wybór programatora ST Link

W kolejnym okienku wybieramy mikrokontroler, jakim dysponujemy. Ja wybrałem STM32F107VB, bo taki mam na swoim „motylu". W ostatnim oknie wybieramy programator ST-Link, po czym możemy kliknąć na *Finish* (**rysunek 6**).

Po lewej stronie znajduje się drzewo naszego projektu. Z istniejących plików interesują nas znajdujące się w katalogu src (pliki źródłowe) plik main.c i stm32f10x_it.c. W pierwszym będzie znajdować się główna część programu, w drugim procedury obsługi przerwań. Otwieramy najpierw plik main.c. Znajdziemy w nim gotowy szablon projektu, przygotowany na gotowe zestawy ewaluacyjne. Ja proponuję wyczyścić całą zawartość pliku i przygotować go samemu.

Na początku za pomocą *#include* musimy dodać główny plik nagłówkowy:

```
#include "stm32f10x.h"
```

Następnie tworzymy funkcję main:

```
int main(void)
{
}
```

Funkcja main to główna funkcja każdego programu, od niej rozpoczyna się wykonywanie programu. Więc na początek umieścimy tam procedury konfigurujące peryferia mikrokontrolera. Ja procedury konfiguracyjne umieściłem w osobnych funkcjach, które kolejno wywołuje na początku funkcji *main()*. Przed funkcją main należy umieścić deklaracje funkcji:

```
void RCC_Configuration(void);
void GPIO_Configuration(void);
void EXTI_Configuration(void);
void NVIC_Configuration(void);
```

Ciała tych funkcji znajdują się na **listingu 1**. Pierwsza funkcja RCC_Configuration konfiguruje wewnętrzne układy generowania sygnałów zegarowych. Usta-

wiłem je tak by rdzeń mikrokontrolera taktowany był z maksymalną prędkością 72 MHz. Ponieważ na „motylu" znajduje się kwarc 25 MHz, należało użyć obu pętli PLL. Sygnał jest najpierw dzielony przez 5, następnie mnożony w pętli PLL2 razy 8, co daje wartość 40 MHz. Sygnał ten trafia na pętlę PLL1 gdzie jest ponownie dzielony przez pięć i mnożony razy 9, co daje wartość 72 MHz. Po tym zabiegu wystarczy użyć jako źródła sygnału zegarowego wyjścia pętli PLL1. Ponadto w procedurze ustawiane są opóźnienia dla pamięci Flash oraz dzielniki sygnału zegarowego dla magistrali systemowych.

Funkcja GPIO_Configuration konfiguruje GPIO, czyli wejścia/wyjścia mikrokontrolera. W mojej wersji procedury konfigurowany jest port E, do którego podłączony jest joystick i dwie diody LED. Funkcja EXTI_Configuration konfiguruje przerwania pochodzące z GPIO. Wszystkie wejścia współpracujące z joystickiem są ustawione jako źródła przerwania. W dalszej części opiszę jak napisać funkcje obsługi tych przerwań. Funkcja NVIC_Configuration konfiguruje kontroler przerwań NVIC. W tej wersji działanie funkcji sprowadza się do uruchomienia przerwań EXTI, czyli pochodzących z GPIO, do których podłączony jest joystick.

Kolejne zmiany, jakich dokonamy będą obejmowały plik stm32f10x_it.c znajdujący się w tym samym katalogu co nasz plik main.c. W pliku tym umieszczone są funkcje obsługujące przerwania, oczywiście w naszym nowym projekcie są one jeszcze puste. Brakuje tu jednak funkcji obsługujących przerwań z kontrolera EXTI, czyli tych które wywołane zostaną przez joystick. Należy więc dodać do pliku funkcje void EXTI9_5_IRQHandler(void)oraz void EXTI15_10_IRQHandler(void). Pierwsza z funkcji będzie wspólna dla przerwań z pinów 8 i 9 portu GPIOE, a druga dla przerwań z pinów 10, 11 i 12 tego portu. Musi więc istnieć mechanizm, który umożliwi nam sprawdzenie z jakiego pinu pochodzi przerwanie. Służy do tego funkcja EXTI_GetITStatus(). Przykładowo, aby sprawdzić czy przerwanie wywołał pin 8 stworzymy następujący warunek:

```
if(EXTI_GetITStatus(EXTI_Line8) != RESET)
```

Należy pamiętać, że sygnały przerwań pochodzące z kontrolera EXTI należy kasować ręcznie, wiec procedura wykonywana w razie zgodności warunku powinna się kończyć wywołaniem funkcji:

```
EXTI_ClearITPendingBit(EXTI_Line10);
```

Gotowe funkcje obsługi kontrolera EXTI przedstawia **listing 2**.

Kompilowanie i uruchomienie programu

W tym momencie mamy gotowy szablon projektu, który możemy zacząć wypełniać kodem naszej gry. Zanim jednak przystąpimy do pisania gry wypróbujmy nasz szablon, a przy okazji nauczmy się jak kompilować i przenosić program do mikrokontrolera, gdzie zostanie uruchomiony. Aby wykonywanie naszego programu przynosiło jakieś rezultaty dodajmy parę linijek programu. Proponuję sprawdzić działanie przerwań wywołanych naciskaniem joysticka. Otwieramy plik stm32f10x_it.c, odnajdujemy funkcję obsługującą przerwania z pinów 10 do 15 i w warunku sprawdzającym

Listing 1. Funkcja konfigurująca porty I/O oraz sygnały taktujące pracą mikrokontrolera

```
//Funkcje konfiguracyjne
//Funkcja konfigurująca porty GPIO
void GPIO_Configuration(void)
{
//inicjalizacja struktury konfiguracji GPIO
  GPIO_InitTypeDef GPIO_InitStructure;
//uruchomienie taktowania dla portu GPIOE
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOE | RCC_APB2Periph_
AFIO, ENABLE);
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_14 | GPIO_Pin_15;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
//ustawienie pinów 14 i 15 portu GPIOE jako wyjść
  GPIO_Init(GPIOE, &GPIO_InitStructure);
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9 | GPIO_
Pin_10 | GPIO_Pin_11 | GPIO_Pin_12;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING  ;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
//ustawienie pinów 8, 9, 10, 11 i 12 portu GPIOE jako wejść
  GPIO_Init(GPIOE, &GPIO_InitStructure);
}

//Funkcja konfigurująca system sygnałów taktowania
void RCC_Configuration(void)
{
  ErrorStatus HSEStartUpStatus;
  RCC_DeInit();  //zerowanie ustawień RCC
  RCC_HSEConfig(RCC_HSE_ON);  //włącz HSE
//oczekiwanie na gotowość HSE
  HSEStartUpStatus = RCC_WaitForHSEStartUp();
  if (HSEStartUpStatus == SUCCESS)
  {
//włączenie bufora dla pamięci Flash
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);
//ustawienie opóźnień dla pamięci Flash
    FLASH_SetLatency(FLASH_Latency_2);
//taktowanie rdzenia sygnałem 1:1 (72 MHz)
    RCC_HCLKConfig(RCC_SYSCLK_Div1);
//taktowanie magistrali APB2 sygnałem 1:1 (72MHz)
    RCC_PCLK2Config(RCC_HCLK_Div1);
//taktowanie magistrali APB1 sygnałem 1:2 (36 MHz)
    RCC_PCLK1Config(RCC_HCLK_Div2);
//ustawienie PLL2 1:5*8  (25MHz/5*8=20Mhz)
    RCC_PREDIV2Config(RCC_PREDIV2_Div5 );
    RCC_PLL2Config(RCC_PLL2Mul_8);
    RCC_PLL2Cmd(ENABLE);
//oczekiwanie na uruchomienie PLL2
    while(RCC_GetFlagStatus(RCC_FLAG_PLL2RDY) == RESET) {};
//ustawienie PLL1 PLL2CLK/5*9 (20MHz/5*9=72MHz)
    RCC_PREDIV1Config(RCC_PREDIV1_Source_PLL2,RCC_PREDIV1_Div5);
    RCC_PLLConfig(RCC_PLLSource_PREDIV1, RCC_PLLMul_9);
    RCC_PLLCmd (ENABLE);
//oczekiwanie na uruchomienie PLL1
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET) {}
//ustawienie pętli PLL1 jako źródła sygnału zegarowego dla systemu
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);
//oczekiwanie aż pętla PLL1 stanie się źródłem sygnału zegarowego
  while (RCC_GetSYSCLKSource() != 0x08) {}
  }
}

//Funkcja konfigurująca GPIO jako źródła przerwań (kontroler EXTI)
void EXTI_Configuration(void)
{
//inicjalizacja struktury konfiguracji przerwań EXTI
  EXTI_InitTypeDef EXTI_InitStructure;
//aktywowanie funkcji EXTI pinów joystick'a
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource8);
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource9);
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource10);
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource11);
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOE, GPIO_PinSource12);
  EXTI_InitStructure.EXTI_Line = EXTI_Line8 | EXTI_Line9 | EXTI_
Line10 | EXTI_Line11 | EXTI_Line12;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
//przerwanie przy zboczu opadającym
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
//uaktywnienie kontrolera EXTI
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);
}

//Funkcja konfigurująca kontroler przerwań NVIC
void NVIC_Configuration(void)
{
//inicjalizacja struktury konfiguracji kontrolera przerwań
  NVIC_InitTypeDef NVIC_InitStructure;
//wybór grupy priorytetów
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
//konfiguracja przerwania od GPIO piny od 5 do 9
  NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
//konfiguracja przerwania od GPIO piny od 10 do 15
  NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
```

Listing 2.
```
//Obsługa przerwania od przycisków joystick'a, linie: GPIOE.8 i GPIOE.9
void EXTI9_5_IRQHandler(void)
{
  //Obsługa przerwania z GPIOE.8 - przycisk OK
  if(EXTI_GetITStatus(EXTI_Line8) != RESET)                  //sprawdź z którego pinu pochodzi przerwanie
  {
    EXTI_ClearITPendingBit(EXTI_Line8);                      //zeruj źródło przerwania
  }
  //Obsługa przerwania z GPIOE.8 - przycisk w górę
  if(EXTI_GetITStatus(EXTI_Line9) != RESET)                  //sprawdź z którego pinu pochodzi przerwanie
  {
    EXTI_ClearITPendingBit(EXTI_Line9);                      //zeruj źródło przerwania
  }
}
//Obsługa przerwania od przycisków joystick'a, linie: GPIOE.10, GPIOE.11 i GPIOE.12
void EXTI15_10_IRQHandler(void)
{
  extern uint8_t stan_gry;
  //Obsługa przerwania z GPIOE.10 - przycisk w dół
  if(EXTI_GetITStatus(EXTI_Line10) != RESET)                 //sprawdź z którego pinu pochodzi przerwanie
  {
    EXTI_ClearITPendingBit(EXTI_Line10);                     //zeruj źródło przerwania
  }
  //Obsługa przerwania z GPIOE.11 - przycisk w prawo
  else if(EXTI_GetITStatus(EXTI_Line11) != RESET)            //sprawdź z którego pinu pochodzi przerwanie
  {
    EXTI_ClearITPendingBit(EXTI_Line11);                     //zeruj źródło przerwania
  }
  //Obsługa przerwania z GPIOE.12 - przycisk w lewo
  else if(EXTI_GetITStatus(EXTI_Line12) != RESET)            //sprawdź z którego pinu pochodzi przerwanie
  {
    EXTI_ClearITPendingBit(EXTI_Line12);                     //zeruj źródło przerwania
  }
}
```

czy źródłem przerwania był pin 11 dodajemy następującą linię:

```
GPIO_ResetBits(GPIOE,GPIO_Pin_14);
```

Spowoduje ona zaświecenie się diody LED, która podłączona jest do pinu 14. Następnie do warunku sprawdzającego pin 12 dopisujemy:

```
GPIO_SetBits(GPIOE,GPIO_Pin_14);
```

Jeżeli program będziemy uruchamiać na "motylu" to poruszanie joystick'a w prawo i lewo będzie powodowało zaświecanie i gaszenie lewej diody LED.

Podczas pisania gry planuję wykorzystać timer systemowy SysTick. Wypróbujmy zatem jego działanie. Odnajdźmy w pliku stm32f10x_it.c funkcję void SysTick_Handler(void) i dopiszmy w jej ciele kolejne linijki programu:

```
if(GPIO_ReadInputDataBit(GPIOE, GPIO_
Pin_15))
  GPIO_ResetBits(GPIOE,GPIO_Pin_15);
else
  GPIO_SetBits(GPIOE,GPIO_Pin_15);
```

Funkcja GPIO_ReadInputDataBit(GPIOE, GPIO_Pin_15) sprawdza aktualny stan pinu 15 portu GPIO do którego podłączona jest druga dioda LED „motyla". W zależności od tego czy dioda jest już zaświecona czy zgaszona, kod ten będzie powodował jej zgaszenie bądź zaświecenie. Teraz należy skonfigurować sam timer, do czego wystarczy wywołanie zaledwie dwóch funkcji:

```
#define SysTick_Freq 9000000
SysTick_Config(SysTick_Freq/2);
SysTick_CLKSourceConfig(SysTick_CLKSource_
HCLK_Div8);
```

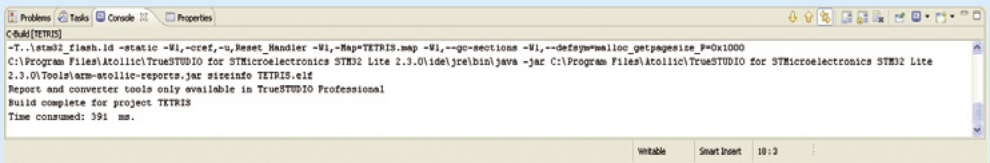
Funcja SysTick_CLKSourceConfig ustawia jedno z dwóch źródeł taktowania timera SysTick. Wywołanie jej z argumentem SysTick_CLKSource_HCLK_Div8 spowoduje taktowanie timera zegarem podzielonym przez 8. W naszym przypadku będzie to 72Mhz/8=9MHz. Druga z funkcji ustawia wartość, od jakiej zacznie odliczanie timer. My zadeklarowaliśmy stałą równą 9000000 odpowiadającą częstotliwości 9 MHz i używamy jej podzielonej przez 2 jako argumentu dla tej funkcji. Da nam to częstotliwość wywołań przerwania SysTick dwóch razy na sekundę, co powinno spowodować zaświecenie się drugiej diody LED raz na sekundę.

Kompilowanie projektu przeprowadzamy naciskając przycisk z symbolem młotka na pasku zadań. Jeżeli proces kompilowania przebiegł pomyślnie, powinna ona zakończyć się pozytywnym komunikatem w oknie konsoli znajdującej się w dolnej części okna programu (**rysunek 7**).

Sprawdźmy działanie naszego programu, kopiując go do pamięci procesora. Najpierw należy podłączyć programator do płytki „motyla" następnie używając dwóch kabli USB podłączyć płytkę i programator do komputera. Oznaką, że programator został podłączony będzie pojawienie się katalogu przenośnego dysku USB z trzema plikami dokumentacji. Teraz naciskamy małą czarną strzałkę obok ikonki małego zielonego żuczka. Z rozwiniętego w ten sposób menu wybieramy Debug Configurations…, otwiera się okno konfiguracji debugera. Wybieramy zakładkę Debugger i upewniamy się czy JTAG probe ustawiony jest na ST-Link, a interfejs na JTAG:.

Następnie naciskamy przycisk Debug na dole okna. Jeżeli program poprawnie połączy się z mikrokontrolerem to powinna zmienić się perspektywa naszego środowiska na Debug:.

W tym momencie nasz program załadowany jest już do pamięci Flash mikrokontrolera, jednak debuger wstrzymał jego wykonywanie. Aby uruchomić program naciskamy zielona strzałkę resume na pasku narzędzi lub naciskamy F8. Jeżeli wykonaliśmy prawidłowo wszystkie kroki efekt powinien być widoczny w postaci migającej diody „motyla". Zaś poruszanie joystickiem z prawo i w lewo będzie zaświecać i gasić drugą diodę LED.



Rysunek 7. Komunikat o prawidłowym przebiegu kompilowania programu

Rafał Kędzierski