Sterowniki robotów

Laboratorium – Debugowanie

Zaawansowane techniki debugowania

Wojciech Domski

Dokument `sr_lab02.pdf`, Wersja 2.0.0

Spis treści

1 Wprowadzenie 2
2 Opis ćwiczenia 2
3 Narzędzia do debugowania 2
3.1 Debugowanie z wykorzystaniem Atollic . . . 3
3.2 Uruchomienie sesji debugera . . . 4
3.3 Wykonywanie krokowe programu . . . 6
3.4 Podgląd wartości zmiennych i wyrażeń . . . 6
3.5 Podgląd rejestrów należących do peryferiów mikrokontrolera . . . 7
3.6 Zamykanie oraz wznawianie sesji . . . 8
3.6.1 Ponowne uruchamianie debuggera . . . 9
3.7 Wgrywanie oprogramowania na mikrokontroler za pomocą Atollic . . . 9
4 USART – Interfejs szeregowy 11
4.1 USART w bibliotece HAL . . . 14
5 SWV – Szeregowy interfejs podglądu 16
6 Przekierowanie funkcji printf() 19
7 STMStudio – wizualizacja danych 20
8 Zadania do wykonania 26
8.1 Dodawanie breakpointów . . . 26
8.2 Poruszanie się po programie . . . 26
8.3 Podgląd zmiennych . . . 26
8.4 Podgląd rejestrów . . . 26
8.5 Przekierowanie funkcji printf() z wykorzystanie interfejsu USART . . . 26
8.6 Przekierowanie funkcji printf() z wykorzystanie interfejsu SWV . . . 28
8.7 STMStudio – rysowanie zawartości zmiennych na ekranie . . . 28
8.8 Uporządkowanie stanowiska . . . 28
9 Podsumowanie 28
Literatura 29

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, edu.domski.pl

1 Wprowadzenie

Debugowanie to proces pozwalający na wykrycie problemów z oprogramowaniem przy użyciu odpowiednich narzędzi. W odróżnieniu do debugowania oprogramowania przeznaczonego na komputery PC, czy też urządzenia mobilne np. z systemem Android proces usuwania błędów na mikrokontrolerze jest operacją, która wymaga odmiennego podejścia. Można wyróżnić cztery zasadnicze komponenty, które są integralną częścią procesu debugowania:

- system wbudowany (*target*) – urządzenie, mikrokontroler, na którym uruchomiony jest program,
- *debugger* – zewnętrzne urządzenie fizyczne dołączane do mikrokontrolera w celu śledzenia wykonywanego programu, a równie często wyposażone w programator (np. ST-Link-V2 [8], [9]),
- *debugger* – aplikacja dostarczająca interfejs programowy do programatora, pozwalająca na sterowanie jego pracą (np. OpenOCD [1]),
- IDE (*Integrated Development Environment*) – zintegrowane środowisko programistyczne pozwalające między innymi na budowanie projektu, czy też jego debugowanie (np. Atollic [2]).

2 Opis ćwiczenia

Ćwiczenie ma na celu zaprezentowanie kilku różnych technik debugowania z wykorzystaniem środowiska Atollic, a także STMStudio [10], [6].

3 Narzędzia do debugowania

Debugowanie oprogramowania dla systemów wbudowanych jest bardzo ważną umiejętnością w procesie wytwarzania oprogramowania. W odróżnieniu od technik, które są obecne w przypadku tworzenia kodu na komputery wyposażone w system operacyjny, jak Linux, czy Windows, to wykrywanie błędów w oprogramowaniu działającym na systemach wbudowanych jest zasadniczo inne. Jednakże koncepcja podstawowej techniki, którą jest krokowe wykonywanie programu jest taka sama. Systemy wbudowane charakteryzują się tym, że nie tylko rozwijamy oprogramowanie dla aplikacji końcowej, ale również musimy bezpośrednio komunikować się z samą warstwą sprzętową co w efekcie finalnym powoduje, że musimy się również zatroszczyć o poprawne działanie całego systemu.

Przykładem, na którym zostanie wykonane śledzenie kodu będzie program mrugającej diody ze względu na niezbyt rozbudowany kod i wyeksponowanie wszystkich niezbędnych aspektów usuwania i znajdowania problemów w wytworzonym oprogramowaniu. Należy wygenerować kod programu za pomocą `STM32CubeMX` [7] dla płytki deweloperskiej `NUCLEO-L476RG`, tak jak to zostało pokazane w [5]. W głównej pętli programu (`while`) należy umieścić poniższy kod:

```
1 HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
2 HAL_Delay(1000);
3 foo();
```

W programie należy również zadeklarować dwie zmienne (*lsb* i *msb*) typu *uint32_t*, jako zmienne globalne. Ponadto, w kodzie powinna znaleźć się również definicja funkcji `foo`, która ma następującą postać:

```
1 void foo( void){
2   int local_variable;
3   ++lsb;
4   if(lsb > 4){
5     lsb = 0;
6     ++msb;
7   }
8 }
```

Program ma na celu co jedną sekundę inkrementować zmienną *lsb* natomiast co 5 sekund zmienna ta jest zerowana, a jednocześnie *msb* jest inkrementowana.

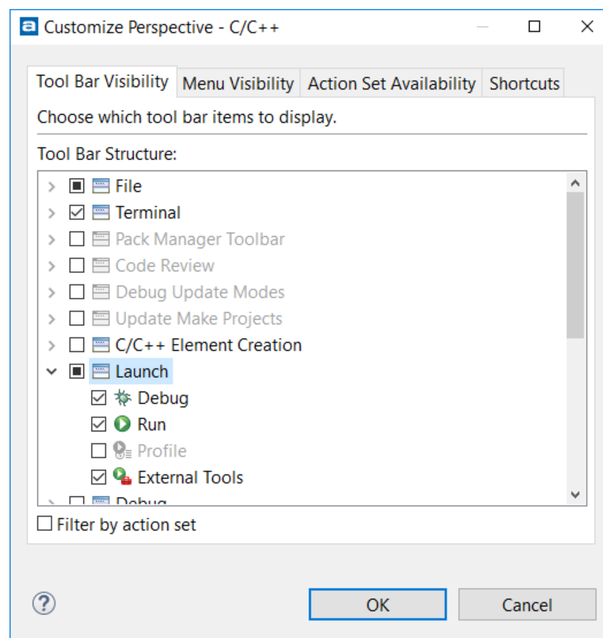
Uwaga! Program powinien kompilować się bez żadnych ostrzeżeń oraz błędów.

Uwaga! Pamiętaj, aby wyłączyć optymalizację kodu (-O0), a także wykorzystać kompilację równoległą (*parallel*) np. -j8 [5]. Ponadto, ustaw automatyczny zapis przed wykonaniem kompilacji, aby uniknąć problemów związanych z rozbieżnością kodu, a plikiem wynikowym wgranym na mikrokontroler. Pamiętaj również, że gdy powtórnie generujesz projekt może okazać się konieczne jego wyczyszcznie jak i przebudowanie indeksu. W tym celu rozwiń menu kontekstowe dla projektu i wybierz *Clean project*, a nastepnie powtórz operację oraz wybierz *Index* → *Rebuild*.

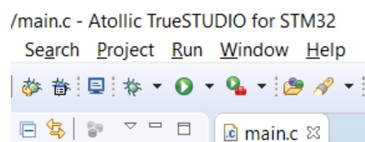
Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, edu.domski.pl

3.1 Debugowanie z wykorzystaniem Atollic

W pierwszej kolejności należy dodać pasek ikon, który zapewni szybki dostęp do uruchamiania procesu debugowania. Można to uczynić przez wybranie z menu programu *Window* → *Perspective* → *Customize Perspective ...*. Ukaże się okno podobne do tego na Rys. 1. Należy zaznaczyć opcję *Launch*, a następnie zatwierdzić wybór przyciskiem *Ok*. Po tej operacji na pasku ikon pojawi się nowa grupa poleceń przydatnych podczas uruchamiania debugera (Rys. 2).

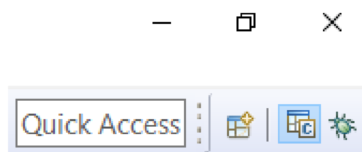


Rysunek 1: Konfiguracja perspektywy dla środowiska Atollic TrueSTUDIO



Rysunek 2: Zestaw ikon stowarzyszonych z akcjami uruchamiania

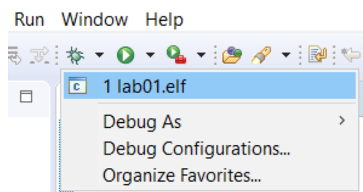
Aby rozpocząć pierwszą sesję debugowania należy nacisnąć ikonę przedstawiającą małą pluskwę (ang. *bug*) na pasku narzędzi, bądź wcisnąć klawisz *F11*. Spowoduje to uruchomienie sesji, która wiąże się z kilkoma etapami. Na początku zostanie sprawdzone, czy projekt posiada jakieś zmiany. Jeśli wykryto zmiany, to wówczas kod źródłowy zostanie skompilowany. Następnym krokiem jest uruchomienie debugera i wgranie wygenerowanego pliku binarnego do mikrokontrolera. Na samym końcu zostanie zmieniona perspektywa, na taką, która jest wyposażona w inne okna przydatne podczas śledzenia wykonywanego kodu. Może się zdarzyć, że po uruchomieniu debuggera nie zmieni się perspektywa na *Debug*. Wówczas należy wybrać, ją ręcznie (*Window* → *Perspective* → *Open Perspective* → *Debug*) lub wykorzystać zestaw ikon umiejscowionych w prawym górnym rogu aplikacji (Rys. 3). W przypadku monitu firewalla odnośnie aplikacji *OpenOCD* należy go zignorować i pozwolić na dalszą pracę *OpenOCD* ponieważ jest to serwer debugera, za pomocą którego środowisko programistyczne komunikuje się z mikrokontrolerem.



Rysunek 3: Zmiana aktualnej perspektywy

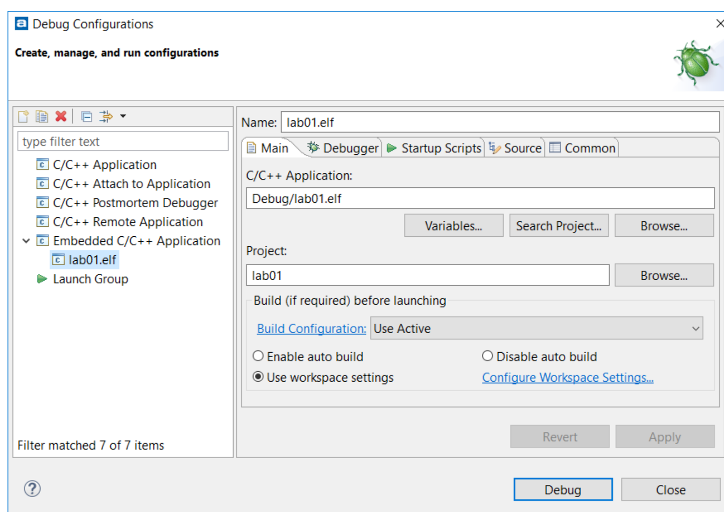
Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, edu.domski.pl

W przypadku, w którym chcemy uruchomić inną konfigurację debugera będzie ona dostępna w małym podmenu dostępnym po naciśnięciu małego trójkąta po prawej stronie od ikony z pluskwą. Wówczas za pomocą menu kontekstowego możemy zarządzać konfiguracjami, uruchomić jedną z już utworzonych (Rys. 4), bądź przejść do tworzenia nowej.



Rysunek 4: Wybór dostępnych konfiguracji startu debugera

Zarządzanie dostępnymi konfiguracjami debugera jest dostępne z poziomu menu kontekstowego opisywanego wyżej. W tym celu należy wybrać opcję *Debug Configurations ...*. Wówczas ukaże się okno podobne do Rys. 5. W tym oknie możemy dodawać, usuwać lub tworzyć nowe konfiguracje, także bazujące na wcześniejszych. Warto podkreślić, że dostępna jest domyślna konfiguracja. Została ona utworzona podczas generowania kodu źródłowego projektu za pomocą programu `STM32CubeMX`. W oknie konfiguracyjnym możliwe jest zarzadzanie jaki sprzetowy debuger będzie wykorzystany, czy ustawienie prędkości transmisji. Do zaawansowanych ustawień należy skrypt startowy dostępny w zakładce *Startup Scripts*, gdzie można ustawić między innymi tryb resetowania mikrokontrolera.



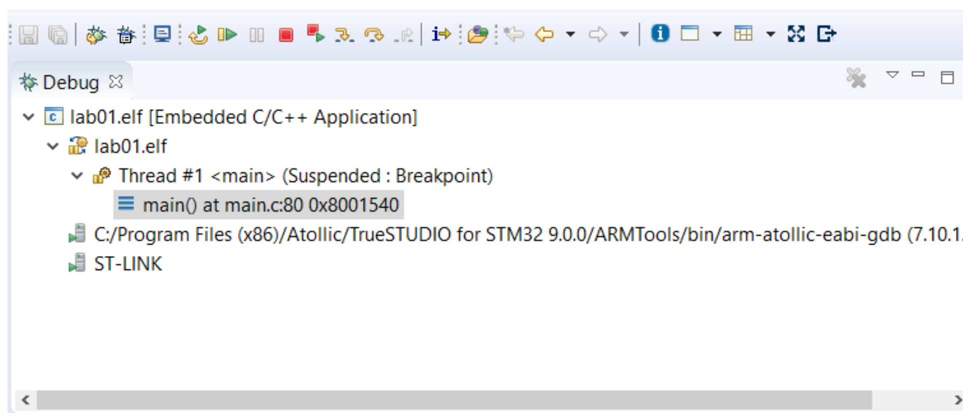
Rysunek 5: Konfiguracja plików startowych debugera

W przypadku zamknięcia okien (eksploratora projektu, outline, Console, Problems, itp.) w obszarze roboczym można przywrócić domyślną konfigurację ułożenia paneli w danej perspektywie za pomocą przywrócenia perspektywy (*Window* → *Perspective Reset Perspective...*). Jest to przydatna opcja podczas, gdy nieumyślnie zostanie ważne okno, a przeszukiwanie Gąszczu dostępnych widoków może czasem okazać się czasochłonne.

3.2 Uruchomienie sesji debugera

Jak już opisano wcześniej po wybraniu akcji debugowania rozpocznie się szereg zadań, które są konieczne do poprawnego uruchomienia sesji debugera. Po przygotowaniu sesji może pojawić się komunikat mówiący o możliwości przełączenia perspektywy. Jedną z własności perspektyw *Debug* jest zakładka o tej samej nazwie *Debug* przedstawiona na Rys. 6. Zawiera ona informację o ostatnich sesjach oraz o ich stanie – aktywnym, bądź zakończonym.

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, `edu.domski.pl`

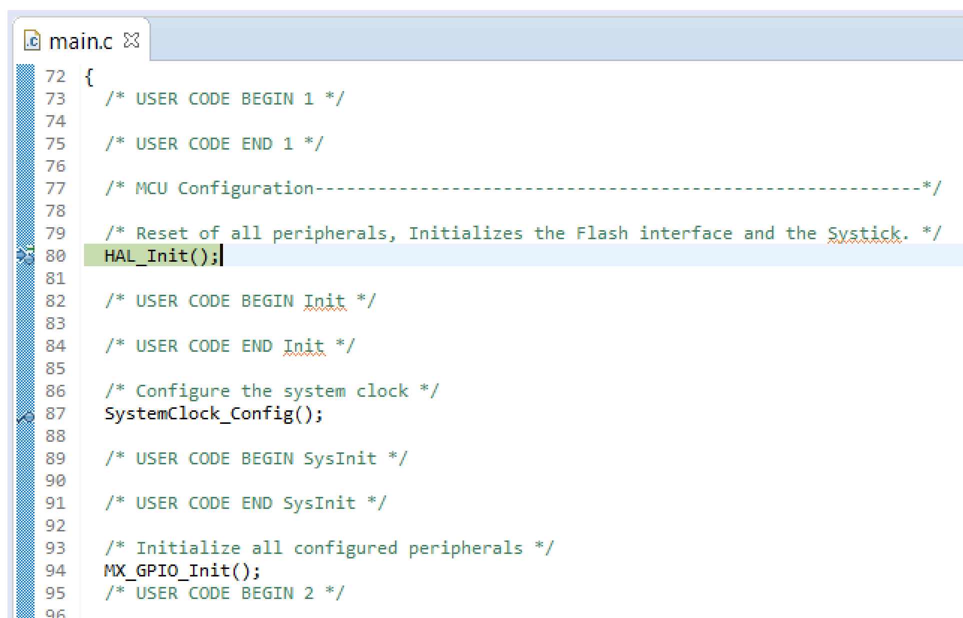


Rysunek 6: Widok zakładki Debug z ostatnimi sesjami

Po pierwszym uruchomieniu debuggera program automatycznie (według domyślnej konfiguracji) jest w trybie zawieszonym (ang. *Suspended*). Aby wznowić program z menu należy wybrać odpowiednią ikonkę składającą się z trójkąta i pionowej kreski po jego prawej stronie, jak to również zostało pokazane na Rys. 6. Po wybraniu tej opcji program wznowi działanie i będzie normalnie się wykonywał realizując swoje zadanie.

Ważnym elementem usuwania błędów z kodu są punkty zatrzymania w kodzie (ang. *breakpoints*). Domyślnie w programie istnieje jeden taki punkt. Jest on zawsze ustawiany na pierwszą instrukcję w funkcji `main()` (Rys. 7). Sam breakpoint w programie przedstawiany jest, jako mały punkt, który widoczny jest również na rysunku obok linii, która zawiera wywołanie funkcji `HAL_Init()`. Ponadto, na Rys. 7 widać jeszcze jeden istotny element, którym jest mała strzałka skierowana w prawą stronę. Wskazuje ona linię kodu, na której aktualnie program jest zatrzymany, a dodatkowo ta linia kodu jest podświetlona.

Aby dodać punkt zatrzymania (*breakpoint*) należy wybrać linijkę kodu, a następnie dwukrotnie kliknąć myszką na obszar, w którym ma pojawić się znacznik *breakpointa*, obszarem tym jest niebieski pionowy pasek obok numerów linii. Widoczne jest to również na Rys. 7. Aby usunąć zatrzymanie kodu w danej linijce należy powtórzyć jeszcze raz operację klikając dwukrotnie na ten punkt.



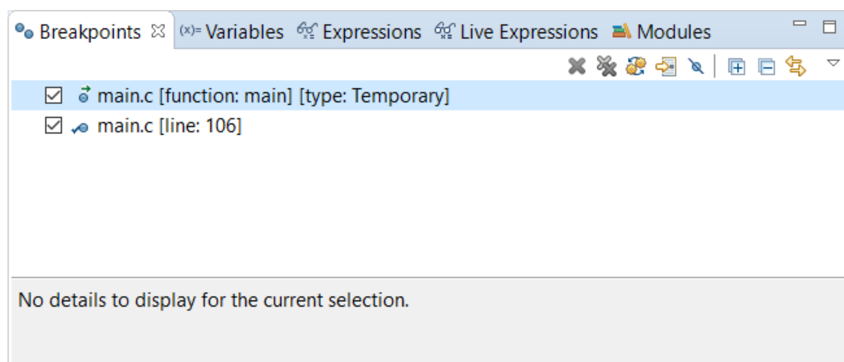
Rysunek 7: Dodawanie punktu zatrzymania w kodzie programu programie

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, edu.domski.pl

3.3 Wykonywanie krokowe programu

Istnieje kilka podstawowych poleceń, które pozwolą na krokowe poruszanie się po programie. Wcześniej nadmienione wznawianie (ang. *Resume*, **F8**), a także wejście do środka funkcji (*Step Into*, **F5**), przeskoczenie do następnej linii programu (*Step Over*, **F6**) oraz zakończenie sesji (*Terminate*, **Ctrl+F2**).

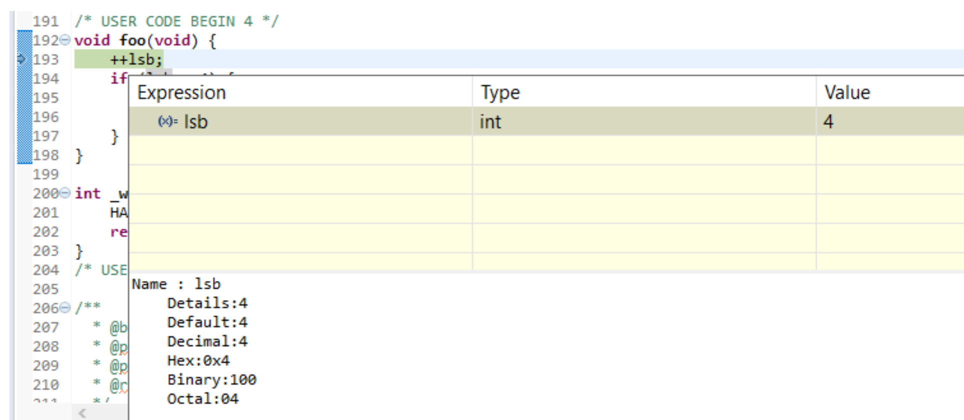
Czasem istnieje konieczność pozostawienia programu, aby ten mógł się wykonywać bez żadnych pułapek. Jednakże jeśli wznowimy wykonywanie programu to wówczas zostanie on zatrzymany na fragmencie programu, gdzie została umieszczona pułapka. Ominięcie pułapek można wykonać na dwa sposoby. Pierwszym z nich jest usunięcie pułapek, na których może nastąpić zatrzymanie wykonywania kodu. Postępowanie to jednak nie jest efektywne, gdyż wymaga ono modyfikacji punktów zatrzymania. Innym znacznie lepszym sposobem jest tymczasowe wyłączanie tychże punktów. Na Rys. 8 zostało przedstawione okno do zarządzania pułapkami. W nim poprzez zaznaczanie pułapek możemy je wyłączać, ale nie usuwać! Pułapka wciąż jest ustawiona, ale nie jest aktywna. Również możliwa jest zmienienie stanu wszystkich pułapek za pomocą ikony z przekreśloną kropką.



Rysunek 8: Zarządzanie breakpointami

3.4 Podgląd wartości zmiennych i wyrażeń

Atollic TrueSTUDIO pozwala na klika trybów podglądu wartości zmiennych. Ważne jest, że dostęp do aktualnej wartości wybranej zmiennej w programie jest dostępny w momencie, kiedy znajduje się on w trybie wstrzymania. Wówczas najeżdżając kursorem na nazwę zmiennej możliwe jest podglądniecie jej aktualnej wartości (Rys. 9).



Rysunek 9: Podglądanie aktualnej wartości zmiennej

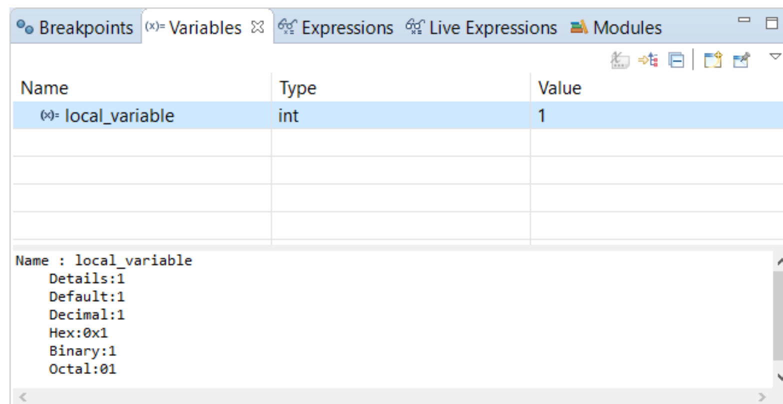
Innym sposobem na podglądanie aktualnego stanu zmiennych jest wykorzystanie zakładki *Variables*, *Expressions*. *Variables* (Rys. 10) daje możliwość podglądu zmiennych lokalnych w danym kontekście w programie oraz ich modyfikacji w trakcie wykonywania kodu. Druga z tych zakładek pozwala nie tylko na podglądanie samych zmiennych, a także obliczanie prostych wyrażeń (Rys. 11). Aby dodać wyrażenie do podglądu należy wykonać jedną z dwóch czynności:

1. Zaznaczyć wyrażenie, a następnie przeciągnąć je do okna.

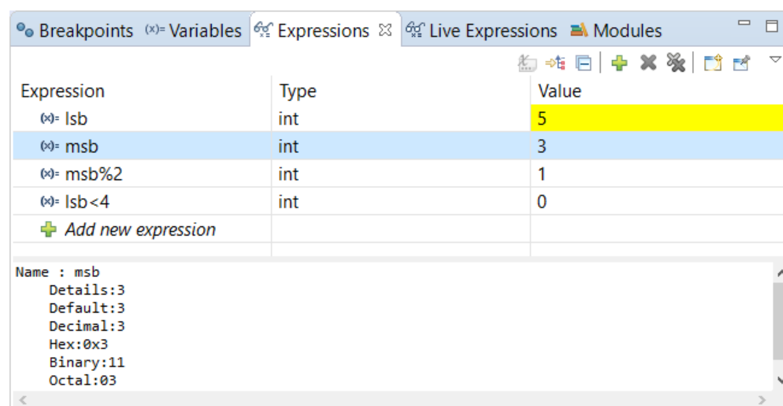
Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, `edu.domski.pl`

2. Wpisać nowe wyrażenie przy pomocy pozycji *Add new expression*.

Na uwagę zasługuje również kolorystyka okna. W przypadku, gdy zmienna jest podświetlana znaczy to, że uległa ona przed chwilą zmianie. Ma to miejsce np. w momencie, w którym mamy ustawiony podgląd zmiennej, a po wykonaniu linii programu w trybie krokowym nastąpi modyfikacja wartości spod pamięci, w którym znajduje się zmienna. Kiedy aktualizowany jest stan wyrażeń?



Rysunek 10: Okno podglądu zmiennych lokalnych



Rysunek 11: Okno podglądu wyrażeń

3.5 Podgląd rejestrów należących do peryferiów mikrokontrolera

Kolejna zakładka, która dostarcza użytecznych informacji na temat peryferiów mikrokontrolera to *SFRs*. Domyślnie nie jest ona aktywna. Wystarczy jednak z menu wybrać *Window* → *Show menu* → *SFRs*, aby pojawiła się ona w aktywnej perspektywie. Zawiera ona aktualny stan rejestrów danego peryferium, jak i samego rdzenia. Aby zażądać aktualizacji informacji o rejestrach danego peryferium, należy je rozwinąć, aby anulować subskrypcję należy je zwinąć. Na rysunkach 13 i 12 zostały przedstawione sytuacje, w której pin wyjściowy jest aktywny (stan wysoki), a w drugim, gdy pin jest nieaktywny (stan niski).

Co można wywnioskować z informacji przedstawionych na rysunkach 12 i 13? Szczegółowy opis rejestrów można znaleźć w nocie aplikacyjnej danego mikrokontrolera.

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, edu.domski.pl

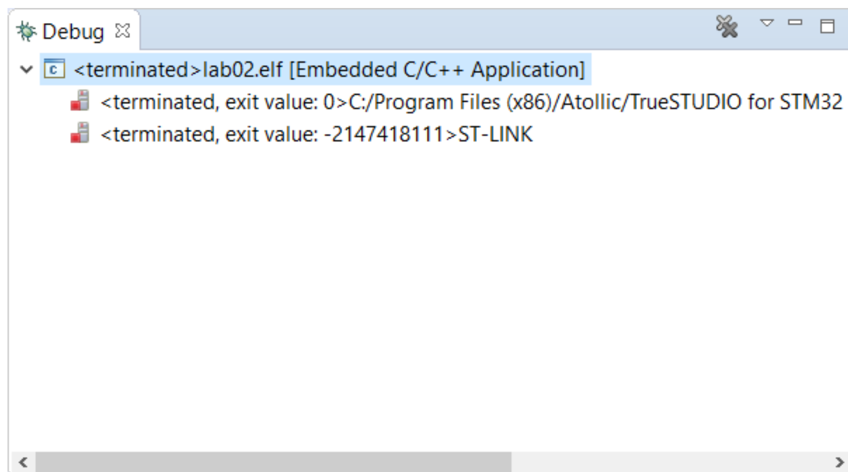
Rysunek 12: Podgląd rejestru GPIO dla portu A

Rysunek 13: Podgląd rejestru GPIO dla portu A

3.6 Zamykanie oraz wznawianie sesji

Po przeprowadzeniu sesji debugowania można ją zakończyć wybierając operację *Terminate* (ikona czerwonego kwadratu) lub wciskając kombinację klawiszy **Ctrl+F2** (Rys. 6). Ważne jest, aby przed rozpoczęciem sesji poprzednia wykorzystywana była już unicestwiona *Terminated*.

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, `edu.domski.pl`



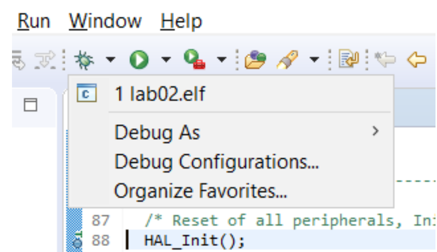
Rysunek 14: Zmiana aktualnej perspektywy

Uwaga! Po zakończeniu debugowania należy zakończyć sesję. Uruchomienie kolejnej sesji może powodować nieoczekiwane błędy, które nie są związane z programem. Aby zweryfikować, czy sesja została zakończona należy odczytać tą informację z zakładki *Debug*. Na rysunku 14 został przedstawiony widok na sesję, która została już zakończona. Jej aktualny status to *Terminated*. W przypadku, gdy istnieje więcej sesji (zakończonych) nie ma to wpływu na poprawne działanie programu.

Warto zwrócić uwagę na możliwość ponownego uruchomienia sesji debugera. W tym celu należy wybrać ikonę z małym czerwonym kwadratem oraz zielonym trójkątem (Rys. 6) . Akcja ta spowoduje, że obecna sesja zostanie zakończona oraz ponownie uruchomiona od nowa. W przypadku, kiedy chcielibyśmy jedynie wysłać sygnał resetu, który spowoduje zresetowanie mikrokontrolera i ponowną jego inicjalizację można wybrać ikonę małego zielonego trójkąta z strzałką (Rys. 6) z poziomu paska narzędzi.

3.6.1 Ponowne uruchamianie debuggera

Jeśli dla projektu została już skonfigurowana sesja debugowania nie należy tworzyć nowych. Aby uruchomić kolejny raz taką sesję można wybrać ją z menu debugowania, gdzie została ona utworzona. Zostało to zaprezentowane na rysunku 15. Równocześnie, jeśli bezpośrednio zostanie naciśnięta ikona małej pluskwy wówczas zostanie uruchomiona sesja, która była użyta jako ostatnia.

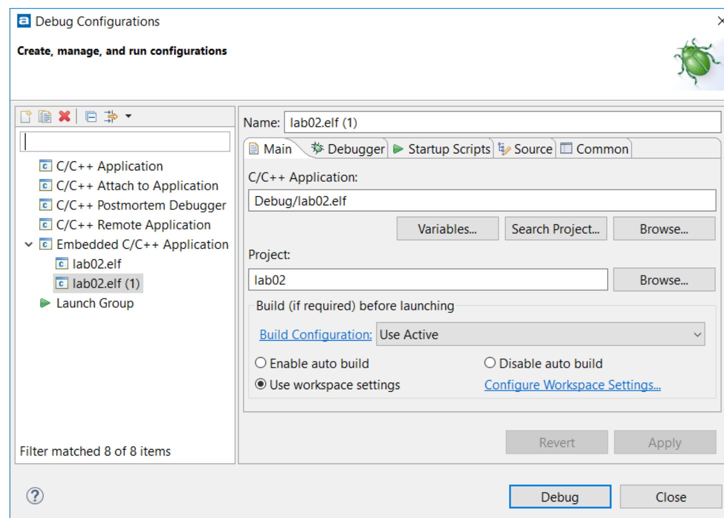


Rysunek 15: Uruchamianie utworzonej sesji

3.7 Wgrywanie oprogramowania na mikrokontroler za pomocą Atollic

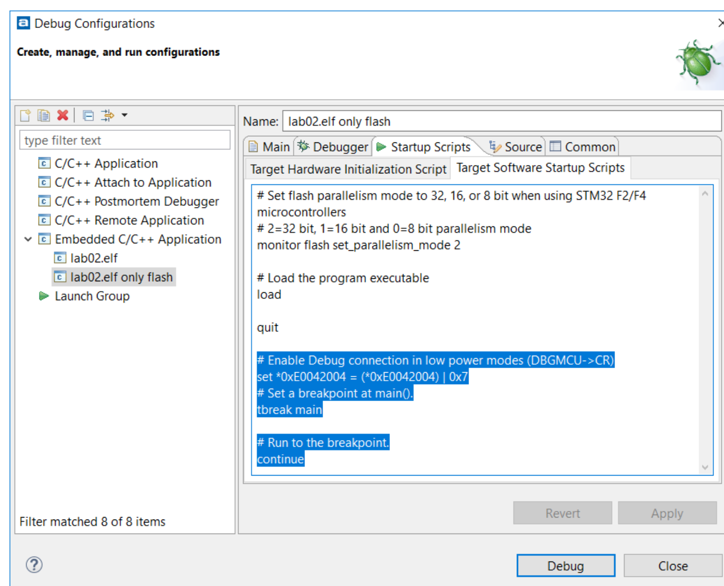
Istnieje kilka metod wgrania najnowszej wersji oprogramowania na mikrokontroler. Jedną z nich jest wykorzystanie narzędzia `ST-Link Utility`, które zostało już opisane wcześniej. Jednakże możliwe jest zaktualizowanie oprogramowania na mikrokontrolerze bezpośrednio przy wykorzystaniu środowiska Atollic TrueSTUDIO. Inną możliwością wgrania nowego oprogramowania na mikrokontroler podczas jest uruchamianie sesji debugowania, jednakże nie zawsze chcemy, aby rozpocząć pełną sesję debugowania. W tym celu należy dodać osobną konfigurację sesji debugowania. Najłatwiejszym sposobem, aby to wykonać jest skopiowanie aktualnej konfiguracji za pomocą *Debug Configurations* (Rys. 5). W oknie konfiguracji zaznaczamy istniejącą konfigurację i wybieramy ikonę tworzenia duplikatu. Efekt można zaobserwować na Rys. 16.

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, edu.domski.pl



Rysunek 16: Tworzenie kopii konfiguracji debugowania

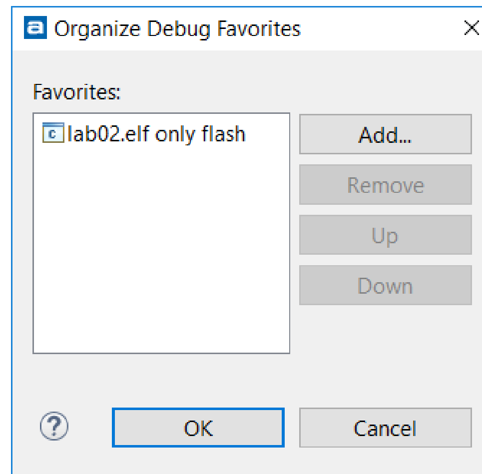
Kolejnym krokiem jest zmodyfikowanie skryptów startowych. W tym celu należy przejść do zakładki *Startup Scripts* → *Target Software Startup Scripts* oraz usunąć wszystko co znajduje się poniżej polecenia *load*, a zaraz za nim dodać polecenie *quit*. Pokazano to na Rys. 17 (zaznaczony fragment przeznaczony jest do usunięcia). Po dokonaniu zmian należy je zapisać wybierając przycisk *Apply*. Kolejnym krokiem jest wybranie przycisku *Debug*. Po tej operacji program zostanie wgrany na mikrokontroler oraz nowa konfiguracja pojawi się w menu obok akcji *Debug*.



Rysunek 17: Tworzenie kopii konfiguracji debugowania

Warto nadmienić, że bez wykonania akcji *Debug* konfiguracja nie pojawi się w dostępnych konfiguracjach do wyboru. Można ją tam również umieścić w inny sposób. Wystarczy wybrać z menu opcję *Organize Favorites ...* i dodać konfigurację do ulubionych (Rys.).

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, edu.domski.pl



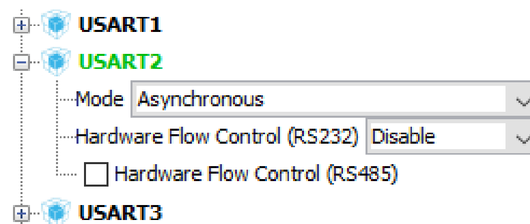
Rysunek 18: Zarządzanie ulubionymi konfiguracjami debugowania

Na tym etapie po uruchomieniu konfiguracji, która została przed chwilą utworzona zostanie stworzona sesja debugowania podczas, której nowe oprogramowanie zostanie przesłane na mikrokontroler. Po zakończeniu operacji przesyłu danych sesja automatycznie zakończy się.

4 USART – Interfejs szeregowy

Interfejs USART jest jednym z najczęściej wykorzystywanych interfejsów komunikacyjnych. Jednocześnie jest on również bardzo prosty w wykorzystaniu. Biblioteka HAL dostarcza wielu funkcji, które pozwalają na jego konfigurację i transmisję danych za pomocą tego interfejsu. Do poprawnej komunikacji wykorzystywana są dwie linie. Jedna linia określana często jako RXD służy do odbierania danych, natomiast TXD do wysyłania danych. Sam interfejs posiada również inne linie, które między innymi służą do kontroli przepływu danych i w efekcie większej kontroli nad samym procesem przesyłu.

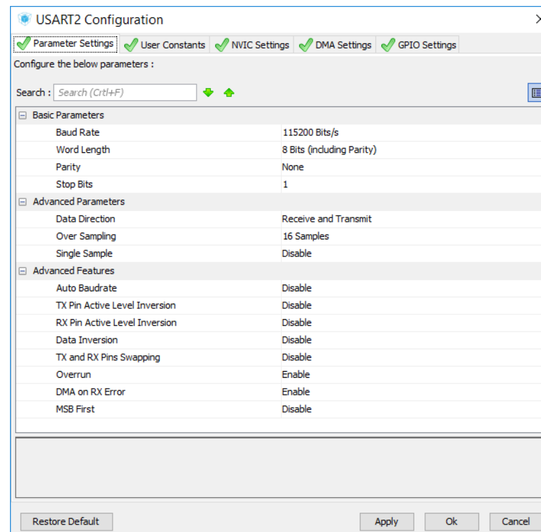
Przejdźmy teraz do omówienia podstawowej konfiguracji z wykorzystaniem narzędzia `STM32CubeMX`. Rys. 19 przedstawia możliwość wyboru pracy. Najczęściej wykorzystywanym trybem pracy jest tryb asynchroniczny i nie wymaga on wykorzystania dodatkowej linii do synchronizowania transmisji.



Rysunek 19: Wybór trybu pracy interfejsu USART

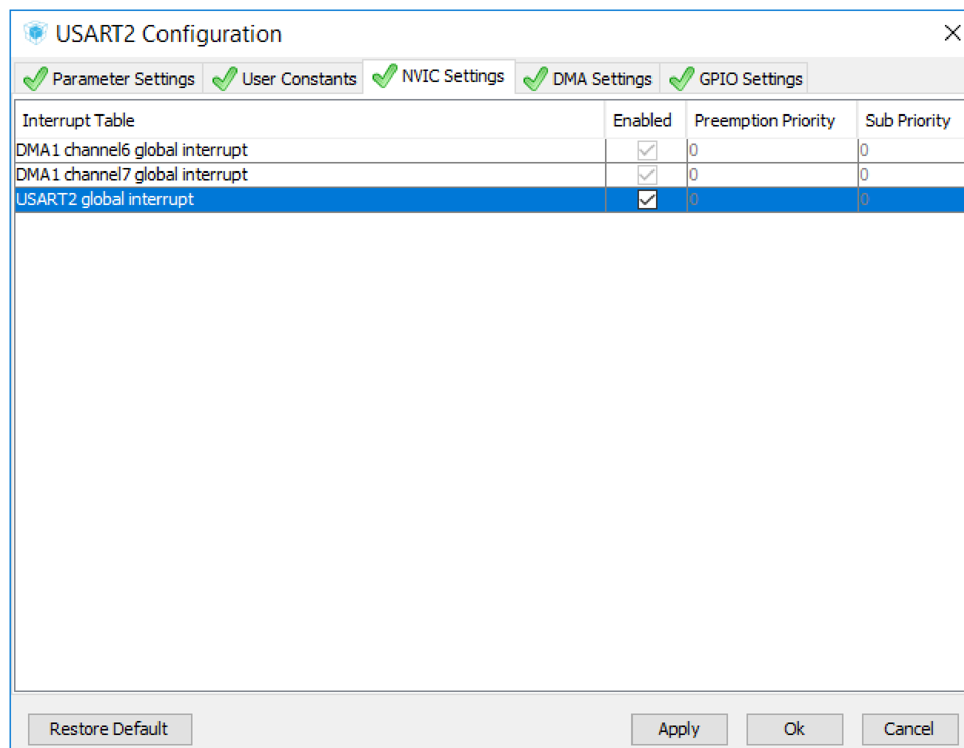
Na Rys. 20 został przedstawiony zrzut ekranu z przykładową konfiguracją USARTa. Należy zwrócić uwagę przede wszystkim na parametry podstawowe (ang. *Basic parameters*). Prędkość transmisji została ustawiona na 115200 $\frac{bit}{s}$, co daje nieco ponad 14 $\frac{kB}{s}$. Długość słowa wynosi 8 bitów wliczając w to bit parzystości. Parzystość może być wyłączona *None* lub ustawiona w trybie parzystości *Even* lub nieparzystości *Odd*. Jest to jedna z form mająca na celu kontrolę poprawności przesyłanych danych. Przykładowo, gdy ustawimy parzystość *Parity* w trybie parzystości *Even* to owy dodatkowy bit będzie miał wartość jeden tylko wtedy, gdy liczba jedynek w przesyłanym słowie jest nieparzysta, tak aby łączna liczba jedynek była już parzysta. Sprawdzaniem poprawności parzystości zajmuje się podsystem w interfejsie szeregowym. W przypadku, gdy parzystość nie będzie się zgadzać to zostaniemy poinformowani o tym fakcie. Warto mieć na uwadze, że uchroni nas to jedynie przed pojedynczym przekłamaniem, ponieważ zamienienie dwóch bitów na przeciwne w słowie będzie przetworzone poprawnie, jednak samo słowo nie będzie już poprawne czego kontrola parzystości nie wykryje. Kolejnym podstawowym parametrem konfiguracyjnym jest liczba bitów stopu. Najczęściej do wyboru mamy, w zależności od interfejsu, mamy 1 lub 2, lecz można wybrać również wartości 0.5 lub 1.5.

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, edu.domski.pl



Rysunek 20: Ustawienia parametrów dla interfejsu USART2

W przypadku interfejsu szeregowego możemy ustawić, czy przerwania od tego peryferium powinny być generowane. Wystarczy zaznaczyć przerwanie jako włączone *Enabled* (Rys. 21).

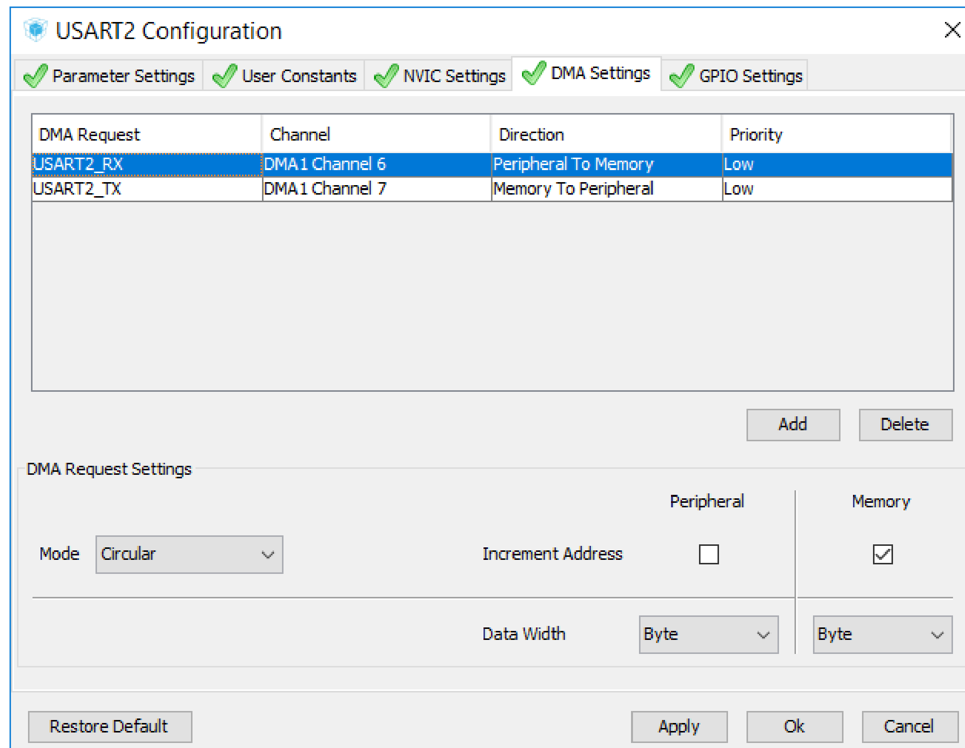


Rysunek 21: Ustawienia przerwań dla interfejsu USART2

`STM32CubeMX` pozwala również na konfigurację kanałów DMA, czyli podsystemu, który pozwala na transakcje związane z przesyłaniem danych bez udziału głównej jednostki obliczeniowej. Na Rys. 22 została przedstawiona konfiguracja dwóch kanałów DMA po jednym dla wysyłania danych i odbierania danych. W obszarze konfiguracyjnym DMA mamy możliwość wybrania trybu pracy DMA (normalny, bądź cyrkularny) oraz sposobu zarządzania danymi po stronie peryferium, jak i pamięci. Zarządzanie danymi pozwala na określenie, czy po każdej transakcji adres z którego porcja danych jest odczytywana, bądź zapisywana ma być inkrementalny, w ten sposób możemy poruszać się liniowo po pamięci. Możemy również określić wielkość przesyłanej porcji danych: pojedynczy bajt, dwa bajty lub cztery. Konfiguracja kanału DMA w trybie cyrkularnym oraz z inkrementalną pamięci jest szczególnie przydatna do

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, `edu.domski.pl`

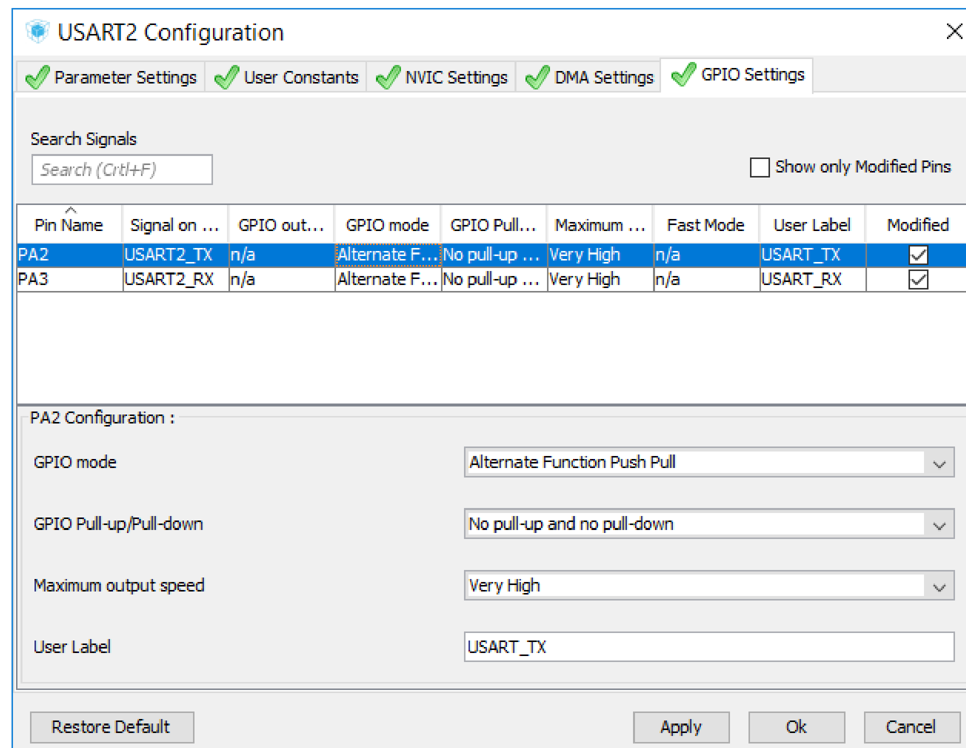
odczytywania danych w sytuacji, w której chcemy zapewnić pewien bufor danych.



Rysunek 22: Ustawienia kontrolera DMA dla interfejsu USART2

Wreszcie na Rys. 23 została przedstawiona konfiguracja portów cyfrowych dla danego peryferium. Warto zwrócić uwagę na to, że tryb GPIO jest ustawiony na alternatywną funkcjonalność *Alternate Function*. Tutaj warto podkreślić, że większość peryferiów dopuszcza mapowanie portów. Oznacza to, że np. linia wysyłająca TXD nie musi być przypisana do portu PA2, a może zostać przypisana do innego portu, ale nie bez ograniczeń. Zazwyczaj mamy do wyboru kilka alternatyw.

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, edu.domski.pl



Rysunek 23: Ustawienia przerwań dla interfejsu USART2

Na tym etapie warto podkreślić dobrą funkcjonalność oprogramowania `STM32CubeMX`. Mianowicie, aplikacja sprawdza, czy przypadkiem podczas konfiguracji peryferium nie wystąpił jakiś błąd. Informacja ta jest dostarczana użytkownikowi na różnych etapach konfiguracji poczawszy od wyboru pinu, ktory będzie podłączony do peryferium, a jest już wykorzystywany przez inne peryferium i nie może być remapowane. W takim przypadku dana funkcjonalnośc zostanie zablokowana, bądź podświetlona na czerwono. Również w obszarze konfiguracji parametrów peryferium następuje walidacja wprowadzanych wartości, bądź wybieranych opcji.

4.1 USART w bibliotece HAL

Wsparcie dla interfejsu szeregowego w bibliotece HAL jest na dobrym poziomie. Do wykorzystania mamy szereg funkcji za pomocą, których możemy odbierać, bądź wysyłać dane w różnych trybach (blokującym, z wykorzystaniem przerwań lub kontrolera DMA).

- `HAL_StatusTypeDef HAL_USART_Transmit (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size, uint32_t Timeout)`,
- `HAL_StatusTypeDef HAL_USART_Receive (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)`,
- `HAL_StatusTypeDef HAL_USART_TransmitReceive (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size, uint32_t Timeout)`.

Powyższe funkcje służą do wysyłania porcji danych, odbierania zadanej liczby danych oraz jednoczesnego wysyłania i odbierania. Są to funkcje blokujące, których zakończenie następuje dopiero w momencie, w którym dane zostały przesłane, bądź upłynął limit czasu wyrażony w milisekundach, a podawanych jako parametr `Timeout`.

- `HAL_StatusTypeDef HAL_USART_Transmit_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size)`,
- `HAL_StatusTypeDef HAL_USART_Receive_IT (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size)`,

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, edu.domski.pl

- HAL_StatusTypeDef HAL_USART_TransmitReceive_IT (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size).

Funkcje przedstawione powyżej odnoszą się również odpowiednio do wysyłania, odbierania oraz jednoczesnego wysyłania i odbierania danych. Istotną różnica jest jednak fakt, że wykorzystywane są przerwania, które informują o zakończeniu danej operacji. Funkcje wykorzystywane do obsługi przerwań są szczególnie przydatne w momencie, w którym program powinien się wykonywać bez przerwy, a jedynie potrzebna nam jest informacja zwrotna, czy została zakończona transmisja. Do prawidłowego działania wymagają one włączenia globalnej obsługi przerwań dla peryferium interfejsu szeregowego. W przeciwnym wypadku nie zostanie uruchomiona obsługa przerwania.

- HAL_StatusTypeDef HAL_USART_Transmit_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint16_t Size),
- HAL_StatusTypeDef HAL_USART_Receive_DMA (USART_HandleTypeDef * husart, uint8_t * pRxData, uint16_t Size),
- HAL_StatusTypeDef HAL_USART_TransmitReceive_DMA (USART_HandleTypeDef * husart, uint8_t * pTxData, uint8_t * pRxData, uint16_t Size).

Podobnie jak w dwóch poprzednich przypadkach i tutaj mamy do czynienia z funkcjami do wysyłania, odbierania oraz wysyłania i odbierania danych w tym samym czasie. Natomiast istotną różnicą jest fakt wykorzystania kontrolera DMA, który pozwoli na odciążenie procesora od operacji kopiowania danych pomiędzy peryferium, a pamięcią.

- HAL_StatusTypeDef HAL_USART_DMAPause (USART_HandleTypeDef * husart),
- HAL_StatusTypeDef HAL_USART_DMAResume (USART_HandleTypeDef * husart),
- HAL_StatusTypeDef HAL_USART_DMAStop (USART_HandleTypeDef * husart).

Wyżej został przedstawiony zestaw funkcji, które wykorzystywane są do kontroli zachowania kontrolera DMA. Pozwalają one na zawieszenie przesyłania danych, jego wznowienie lub całkowite zatrzymanie stowarzyszonego kanału DMA z interfejsem szeregowym.

- void HAL_USART_IRQHandler (USART_HandleTypeDef * husart),
- void HAL_USART_TxCpltCallback (USART_HandleTypeDef * husart),
- void HAL_USART_TxHalfCpltCallback (USART_HandleTypeDef * husart),
- void HAL_USART_RxCpltCallback (USART_HandleTypeDef * husart),
- void HAL_USART_RxHalfCpltCallback (USART_HandleTypeDef * husart),
- void HAL_USART_TxRxCpltCallback (USART_HandleTypeDef * husart),
- void HAL_USART_ErrorCallback (USART_HandleTypeDef * husart).

Biblioteka HAL dostarcza również szereg funkcji, które przede wszystkim dedykowane są obsłudze przerwań od peryferium z szczególnym uwzględnieniem zakończenia transferu danych. Pierwsza z nich jest ogólnym uchwytem przerwania od interfejsu szeregowego i nie zaleca się redefinicji tej funkcji ponieważ spowoduje to, że mechanizm funkcji zwrotnych (ang. *callbacks*) nie będzie działał. Kolejne dwie funkcje wykorzystywane są do informowania, czy nadawanie danych zostało w pełni zakończone, czy jest w połowie. Kolejne dwie funkcje informują o identycznym zdarzeniu, ale w odniesieniu do odbierania danych. Natomiast funkcja `HAL_USART_RxHalfCpltCallback()` jest wywoływana w przypadku, gdy zarowno nadawanie jak i odbiór danych zostało ukończone. W przypadku wystąpienia błędów w transmisji wywoływana jest funkcja `HAL_USART_ErrorCallback()`.

- HAL_USART_StateTypeDef HAL_USART_GetState (USART_HandleTypeDef * husart)
- uint32_t HAL_USART_GetError (USART_HandleTypeDef * husart)

Pierwsza z funkcji wypunktowanych powyżej służy do pobierania informacji o stanie peryferium, czy jest on w gotowości do działania, czy może jest zajęty. Natomiast druga pozwala zidentyfikować źródło problemów z peryferium w przypadku wystąpienia błędu, na przykład, gdy chcemy odebrać dane w trybie blokującym za pomocą funkcji `HAL_USART_Receive()` i funkcja zakończyła się niepowodzeniem. Wówczas wywołanie `HAL_USART_GetError()` może dostarczyć informacji co było przyczyną np. błąd w transmisji wywołany niezgadzającym się bitem parzystości.

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, edu.domski.pl

5 SWV – Szeregowy interfejs podglądu

W skład narzędzi, które wspomagają śledzenie wykonywanego kodu wchodzą:

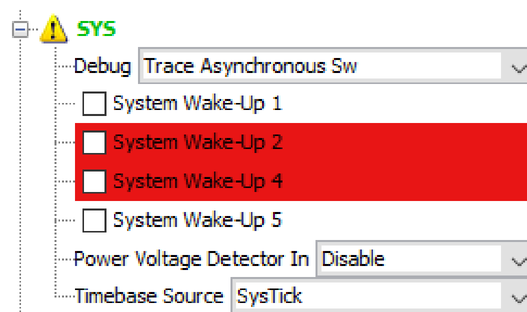
- SWD (ang. *Serial Wire Debuger*),
- SWO (ang. *Serial Wire Output*),
- SWV (ang. *Serial Wire Viewer*),
- ITM (ang. *Instrumentation Trace Macro cell*).

SWD jest nieodłączną częścią każdego mikrokontrolera firmy ST. Jest to interfejs debugera, który przypomina znany inny interfejs – JTAG. W układzie jest on reprezentowany za pomocą dwóch linii SWDIO (linia danych) oraz SWCLK (linia zegara). W rzeczywistości, dzięki aplikacji `STM32CubeMX` możliwe jest skonfigurowanie układu, aby wykorzystywał interfejs JTAG w trybie 4 pinowym, bądź 5 pinowym. Innym narzędziem, które jest wykorzystywane w analizie kodu to SWV. Pozwala ono na śledzenie stanu mikrokontrolera w czasie rzeczywistym. Razem z SWO możliwe jest wyciągnięcie danych na zewnątrz układu w celu ich obserwacji i analizy. SWO jest nichym innym, jak pojedynczym pinem, na którym wystawiane są informacje do świata zewnętrznego. Dzięki tej technologii mamy możliwość obserwowania do 32 niezależnych banków pamięci, a tym samym nadaje się ona bardzo dobrze do przekierowania wyjścia funkcji `printf()`. Samo SWV pozwala na wydobycie następujących danych z układu [3]:

- Okresowe pobieranie wartości licznika wykonywania programu (PC),
- Zdarzenie informujące o dostępnie do pamięci (zarówno zapis, jak i odczyt),
- Zdarzenie informujące o wejściu i opuszczeniu wyjątku,
- Licznika zdarzeń,
- Informacji na temat znaczników czasowych oraz cykli procesora.

Warto powiedzieć kilka słów o ITM. Jest to podsystem, który pozwala na zapis dowolnych danych na wyjście portu SWO. Dzięki czemu przekierowanie wyjścia funkcji `printf()` na port SWO jest możliwe.

Jak już wspomniano jednym z narzędzi, które pozwalają na śledzenie wykonywanego kodu na mikrokontrolerze jest SWV (ang. *Serial Wire Viewer*). Mirokontrolery z rodziny STM32 zostały wyposażone w możliwość asynchronicznego śledzenia stanu układu za pośrednictwem pinu SWO [12], który służy jako wyjście. Wymagane jest tutaj odpowiednie skonfigurowanie portu PB3 za pomocą programu `STM32CubeMX`. Należy z poziomu *Pinout* → *SYS* → *Debug* wybrać opcję *Trace Asynchronous Sw*. Prawidłowa konfiguracja została pokazana na rysunku 24.



Rysunek 24: Konfiguracja interfejsu SWV

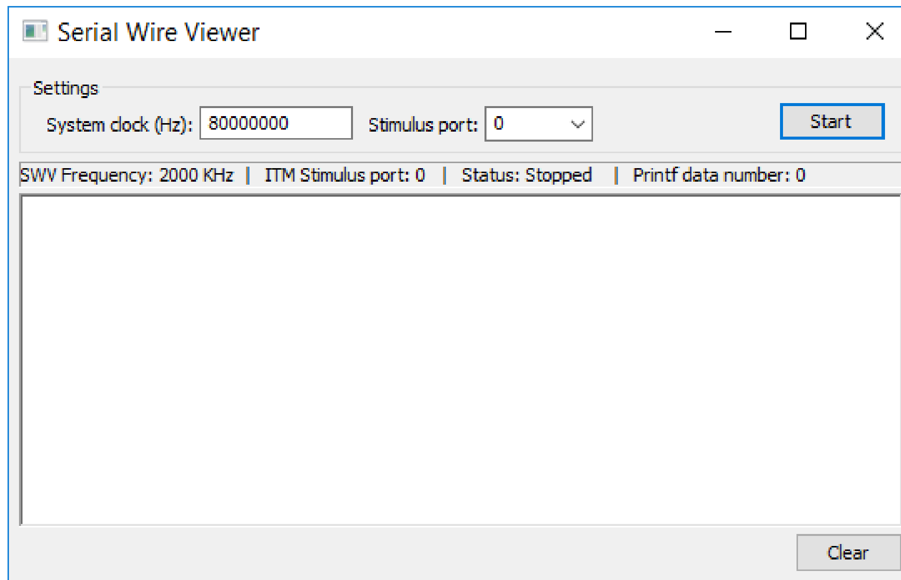
Wywołanie, które spowoduje wysłanie pojedynczego bajtu na wyjście SWO to

```
1 ITM_SendChar(*ptr++);
```

Warto zauważyć, że domyślnie wykorzystywany jest kanał 0 interfejsu ITM. Jest to bardzo istotne podczas konfiguracji nasłuchu. Kolejnym aspektem konfiguracji jest ustawienie odpowiedniej częstotliwość, która powinna być taka sama jak częstotliwość zegara systemowego. Nasłuch można prowadzić za pomocą oprogramowania `ST-Link Utility`, bądź bezpośrednio w środowisku `Atollic TrueSTUDIO`. Aby rozpocząć podgląd wysyłanych danych za pośrednictwem ITM w pierwszej aplikacji należy najpierw

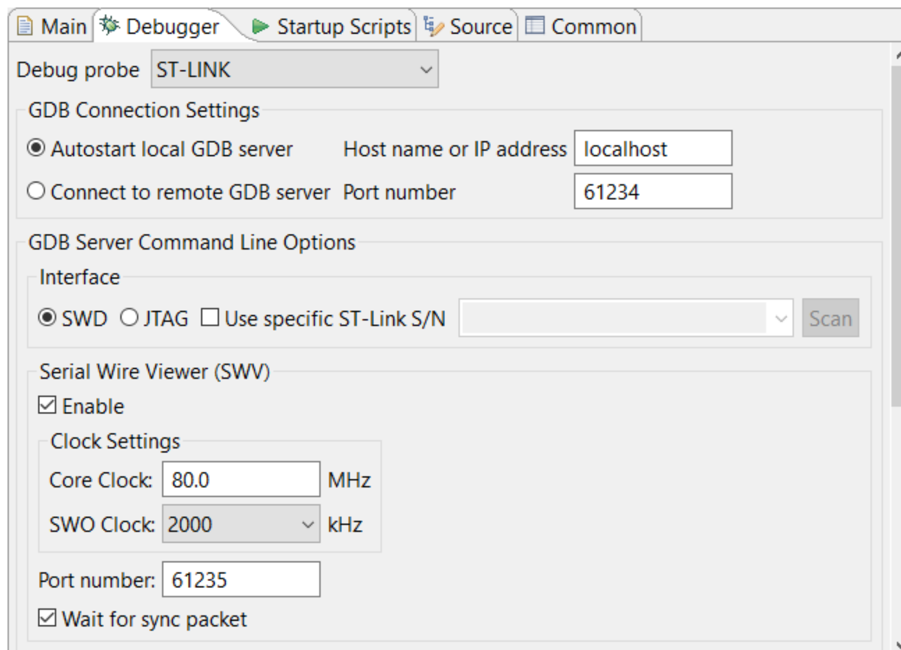
Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, edu.domski.pl

połączyć się z docelowym urządzeniem *Connect*, a następnie z poziomu menu wybrać *ST-LINK* → *Printf via SWO viewer*. Wówczas ukaże się okno, jak na Rys. 25. Tutaj należy ustawić częstotliwość zegara oraz ustawić kanał na 0. Po zakończeniu konfiguracji naciskamy przycisk *Start*; rozpocznie się pobieranie danych z mikrokontrolera.



Rysunek 25: Okno Serial Wire Viewer

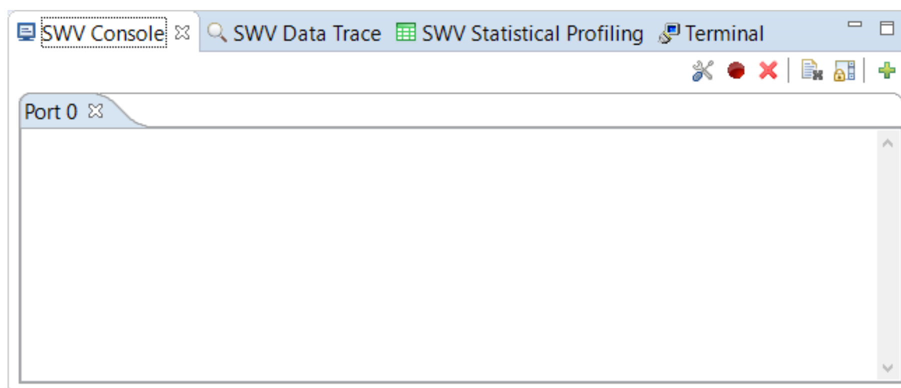
W przypadku środowiska `Atollic TrueSTUDIO` wymagana jest zmiana konfiguracji sesji debugera (Rys. 26).



Rysunek 26: Włączenie interfejsu SWV w konfiguracji sesji debugera

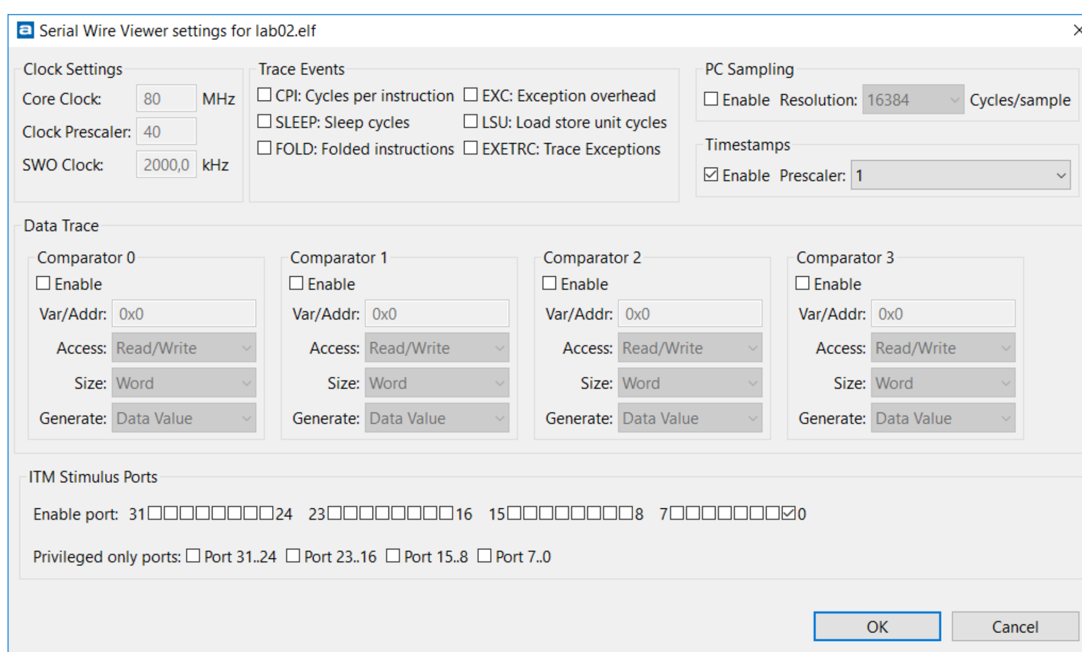
W zakładce *Debugger* należy włączyć *Serial Wire View (SWV)* poprzez zaznaczenie okienka *Enable*. Również tutaj należy ustawić prawidłową częstotliwość rdzenia. Po uruchomieniu debugowania w oknie widoku *SWV Console* należy kliknąć na ikonę klucza z śrubokrętem *Configure trace* (Rys. 27).

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, `edu.domski.pl`



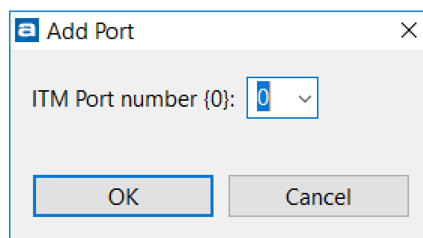
Rysunek 27: Włączenie interfejsu SWV w konfiguracji sesji debugera

Wówczas pojawi się okno konfiguracji SWV (Rys. ??).



Rysunek 28: Okno konfiguracji interfejsu SWV

W dolnej części tego okna należy zaznaczyć, który z kanałów chcemy obserwować (Rys. ??). Po zatwierdzeniu konfiguracji w oknie *SWV Console* należy dodać widok śledzonego kanału. W tym celu należy kliknąć na ikonę zielonego plusa. Ukaże się okno (Rys. 29) z wyborem dostepnych kanałów.



Rysunek 29: Dodawanie kanału interfejsu SWV

Jeśli kanał już został dodany i jest obecny w oknie *SWV Console* to nie będzie on dostępny do wyboru. Po prawidłowym skonfigurowaniu kanału należy nacisnąć czerwoną kropkę *Start Trace* w oknie *SWV Console*. Po wznowieniu wykonywania programu w oknie kanału wewnątrz *SWV Console* zaczną pojawiać się informacje.

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, edu.domski.pl

Podstawowa konfiguracja ogranicza się do wykorzystania kanału numer 0. Poniższe funkcje pozwalają na włączenie dowolnego kanału oraz wysyłanie na wybrany kanał pojedynczego bajtu:

```
void ITM_EnablePort(uint8_t p)
{
  ITM->TER |= 1 << p;
}

__STATIC_INLINE uint32_t ITM_SendCharPort (uint32_t ch, uint8_t p)
{
  if (((ITM->TCR & ITM_TCR_ITMENA_Msk) != 0UL) &&      /* ITM enabled */
  ((ITM->TER & (1UL << p) ) != 0UL)    )       /* ITM Port #p enabled */
  {
    while (ITM->PORT[p].u32 == 0UL)
    {
      __NOP();
    }
    ITM->PORT[p].u8 = (uint8_t)ch;
  }
  return (ch);
}
```

6 Przekierowanie funkcji printf()

Jedną z podstawowych metod związanych z monitorowaniem działania systemu jest logowanie informacji. Do tych celów można wykorzystać funkcję `printf()`, która pozwala na elastyczne dostosowanie komunikatów i wysyłanie ich na standardowe wyjście. W przypadku mikrokontrolerów STM32, jak również innych układów możliwe jest przekierowanie wyjścia funkcji `printf()` na dostępny interfejs np. USART, czy SWV.

Kroki jakie należy wykonać, aby prawidłowo przekierować tekst wyświetlany za pomocą funkcji systemowej `printf()` na wybrany interfejs są następujące:

1. Skonfigurować odpowiedni interfejs w programie `STM32CubeMX`, np. USART.
2. Wygenerować kod z programu `STM32CubeMX`.
3. Dodać odpowiedni nagłówek np. w *main.c* zapewniający wykorzystanie funkcji `printf()`.
4. Wstawić własną definicję funkcji `_write()`.

Nagłówek, jaki należy dołączyć do pliku *main.c* to *stdio.h*. Spowoduje to dodanie odpowiednich prototypów. Również w pliku *main.c* powinna znaleźć się definicja funkcji `_write()`, która zazwyczaj wygląda następująco:

```
int _write(int file, char *ptr, int len) {
  //przepisanie znaku na interfejs

  return len;
}
```

Przykładowa definicja dla portu szeregowego powinna wyglądać podobnie do:

```
int _write(int file, char *ptr, int len) {
  HAL_UART_Transmit(&huart2,ptr,len,50);
  return len;
}
```

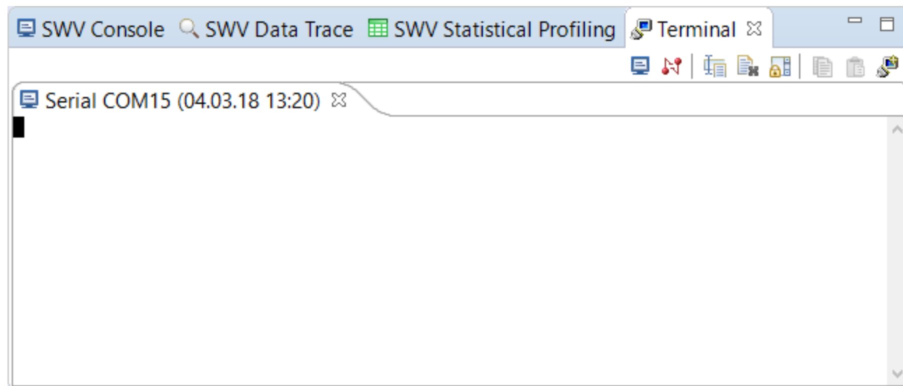
Natomiast w przypadku wykorzystania interfejsu SWV:

```
int _write(int file, char *ptr, int len) {
  int i;
  for (i = 0; i < len; i++) {
    ITM_SendChar(*ptr++);
  }
  return len;
}
```

Uwaga! W przypadku, w którym łańcuch znaków nie jest od razu widoczny na interfejsie należy zakończyć go sekwencją powrotu karetki oraz nowej linii ("\r\n"). Wymusi to natychmiastowe wypisanie tekstu na wybrany interfejs.

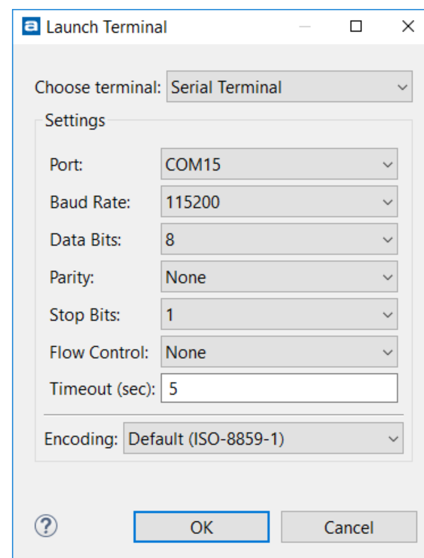
Środowisko `Atollic TrueSTUDIO` pozwala na wykorzystanie wbudowanego narzędzia do śledzenia za pomocą interfejsu SWV, które zostało opisane wcześniej. Pozwala to na łatwe i zintegrowane śledzenie kodu w jednym środowisku. Ponadto, `Atollic TrueSTUDIO` umożliwia również wykorzystanie konsoli *Terminal* (Rys. 30) do podłączenia się do wybranego portu szeregowego.

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, edu.domski.pl



Rysunek 30: Okno terminala w środowisku `Atollic TrueSTUDIO`

Chcąc podłączyć się do interfejsu należy nacisnąć ikonę monitora *Open a Terminal*. W nowym oknie (Rys. 31) można wybrać ustawienia portu szeregowego. Po dodaniu interfejsu w oknie *Terminal* pojawi się podgląd.

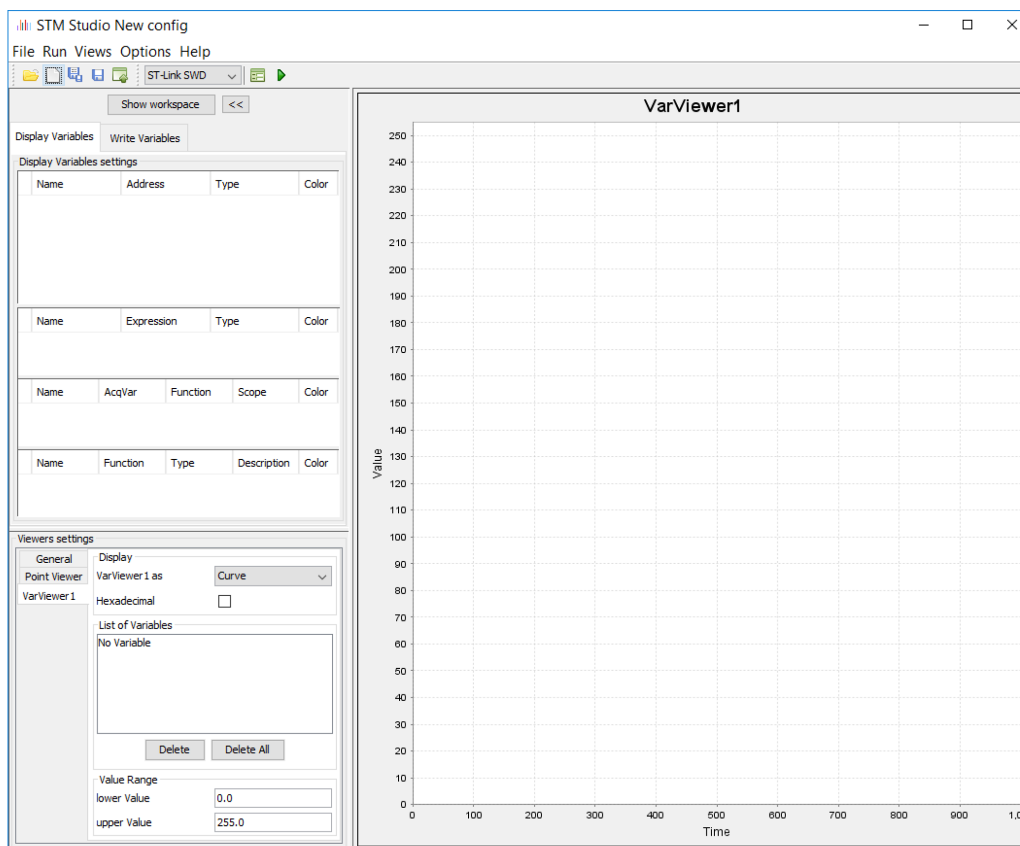


Rysunek 31: Wybór portu szeregowego

7 STMStudio – wizualizacja danych

Kolejnym narzędziem, który nie tylko pozwala na pobieranie informacji o zawartości zmiennych w programie, a także na ich wyświetlanie w formie różnego typu wykresów jest `STMStudio` (Rys. 32). `STMStudio` jest aplikacją prostą w obsłudze, a równocześnie jest bardzo przydatnym narzędziem. Przy niewielkim nakładzie pracy możemy nie tylko podglądać aktualny stan zmiennych w programie, ale także rysować wykresy. To sprawia, że wizualizacje przebiegów czasowych są niezwykle proste do uzyskania i niosą za sobą nieocenione narzędzie od analizy zachowania układu w czasie.

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, `edu.domski.pl`



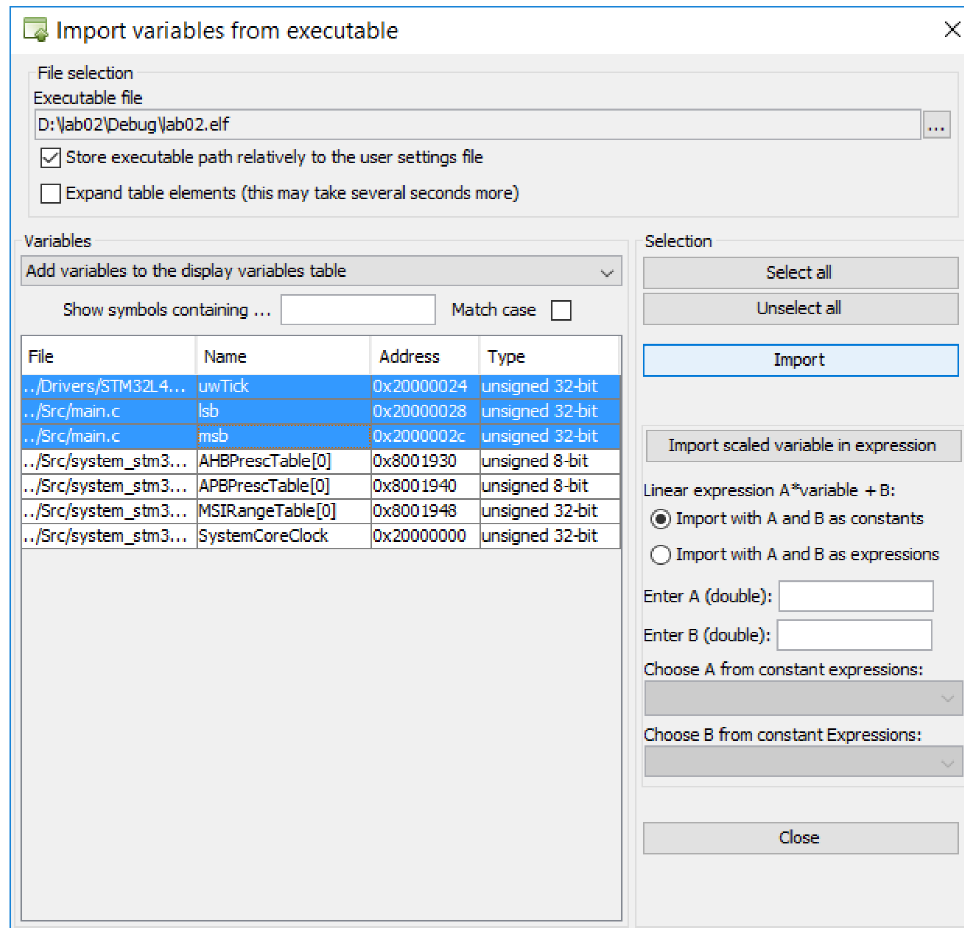
Rysunek 32: Aplikacja STMStudio

Program ten pozwala na dostęp do wszystkich zmiennych, poprzez ich adresy, które przechowywane są w pliku **.elf*. Plik ten generowany jest podczas budowania projektu. Chodzi tutaj o dostęp do obszarów pamięci, które zostały zdefiniowane jako zmienne globalne, gdyż ich położenie w pamięci nie ulega zmianie. Poza odczytem danych z pamięci możliwe jest również ich modyfikowanie co znacząco podnosi walor przedstawianego narzędzia. Przydatne jest to szczególnie w sytuacji, w której wymagana jest zmiana parametrów programu, aby wykonywał on inne zadania. Przykładowo, na mikrokontrolerze został zaimplementowany sterownik PID. Zmiana jego parametrów nawet za pomocą interfejsu może okazać się czasochłonna. Za pomocą bezpośredniego dostępu do pamięci możemy w mgnieniu oka modyfikować nastawy regulatora i od razu obserwować efekty jego pracy.

Podstawowa praca z programem `STMStudio` sprowadza się do następujących czynności:

1. Uruchomienie aplikacji `STMStudio`.
2. Zaimportowanie pliku wykonywalnego **.elf*.
3. Wybór zmiennych, które zostaną zaimportowane.
4. Uruchomienie podglądu.

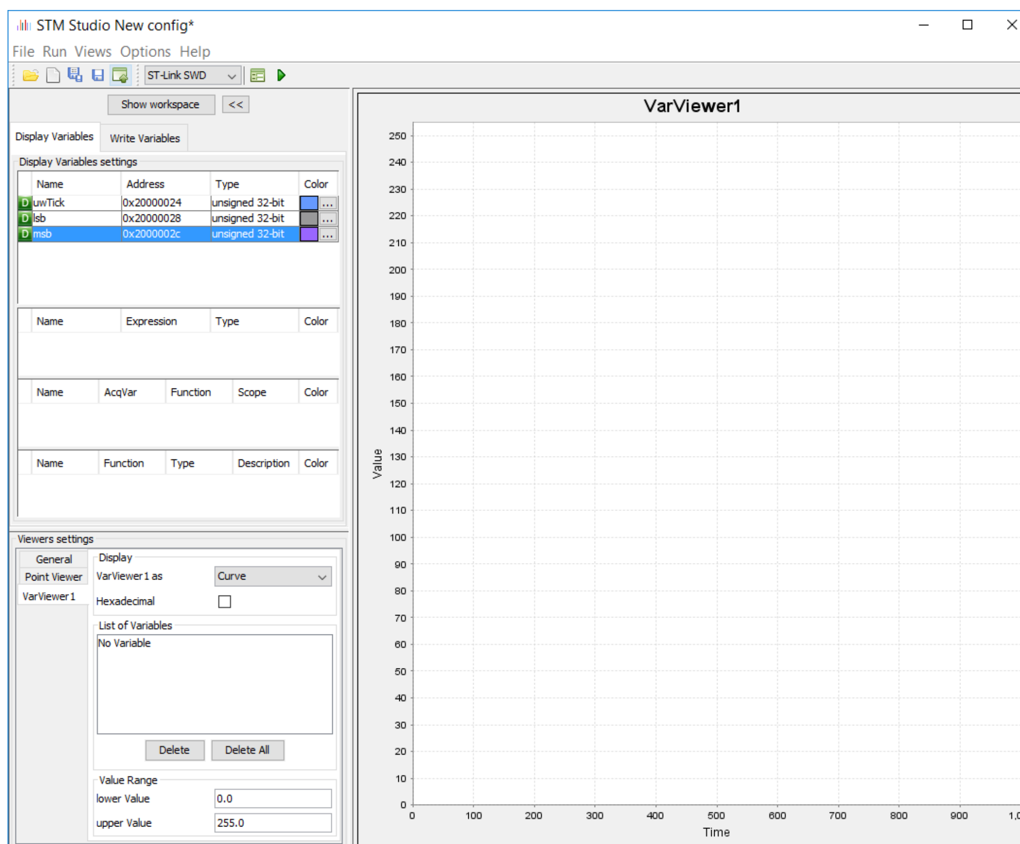
W celu zaimportowania zmiennych do przestrzeni roboczej należy wykorzystać polecenie importu (*File → Import variables*). Ukaże się wówczas okno, jak na Rys. 33.



Rysunek 33: Importowanie zmiennych do programu

Wczytanie pliku wykonywalnego odbywa się przez wskazanie jego położenia, w tym celu należy nacisnąć klawisz ..., który spowoduje otworzenie okna dialogowego w celu wybrania plik **.elf*. Po wybraniu i zaakceptowaniu pliku po chwili w oknie ukażą się zmienne, które można zaimportować. Przykładowo niech będą to zmienne *uwTick*, *lsb* i *msb*. Naciskając przycisk *Import* można je zaimportować do projektu. Po tej operacji w głównym oknie aplikacji pokażą się wybrane zmienne (Rys. 34).

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, edu.domski.pl

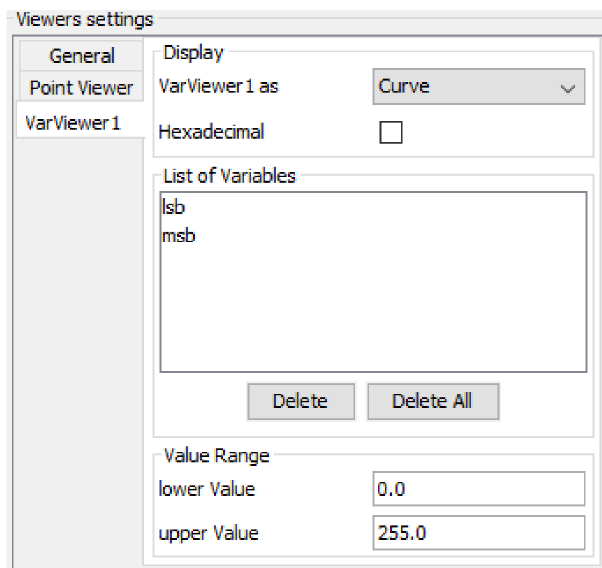


Rysunek 34: Zaimportowane zmienne do aplikacji

Można przystąpić teraz do konfiguracji wykresów. W tym celu należy przeciągnąć zmienne z *Display Variables* → *Display Variables settings* do okna *Viewers settings* → *VarViewer1* → *List of Variables*. `STMStudio` wspiera kilka trybów wyświetlania podglądanych zmiennych, są to:

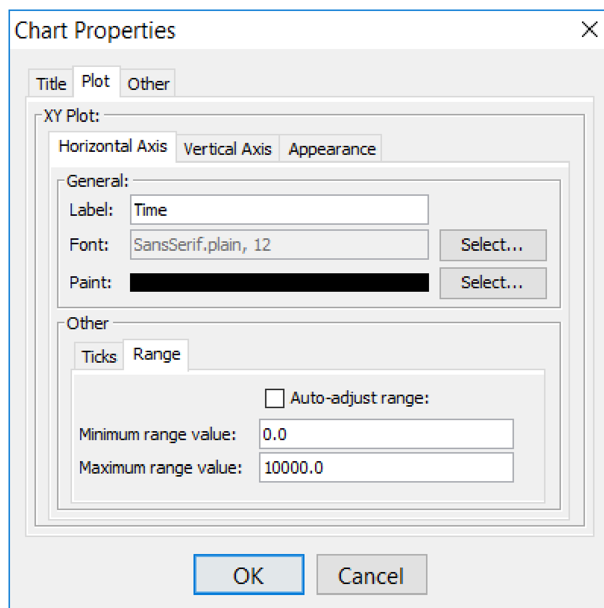
- *Curve* – wykres krzywych w czasie,
- *Bar graph* – wykres słupkowy,
- *Table* – tabela.

Typ wykresu można wybrać w zakładce *Viewers settings* (Rys. 35). W tym samym oknie możemy zmodyfikować minimalną oraz maksymalną wartość wyświetlaną na wykresie za pomocą pól *lower Value* oraz *upper Value*.



Rysunek 35: Wybór ustawień podglądu

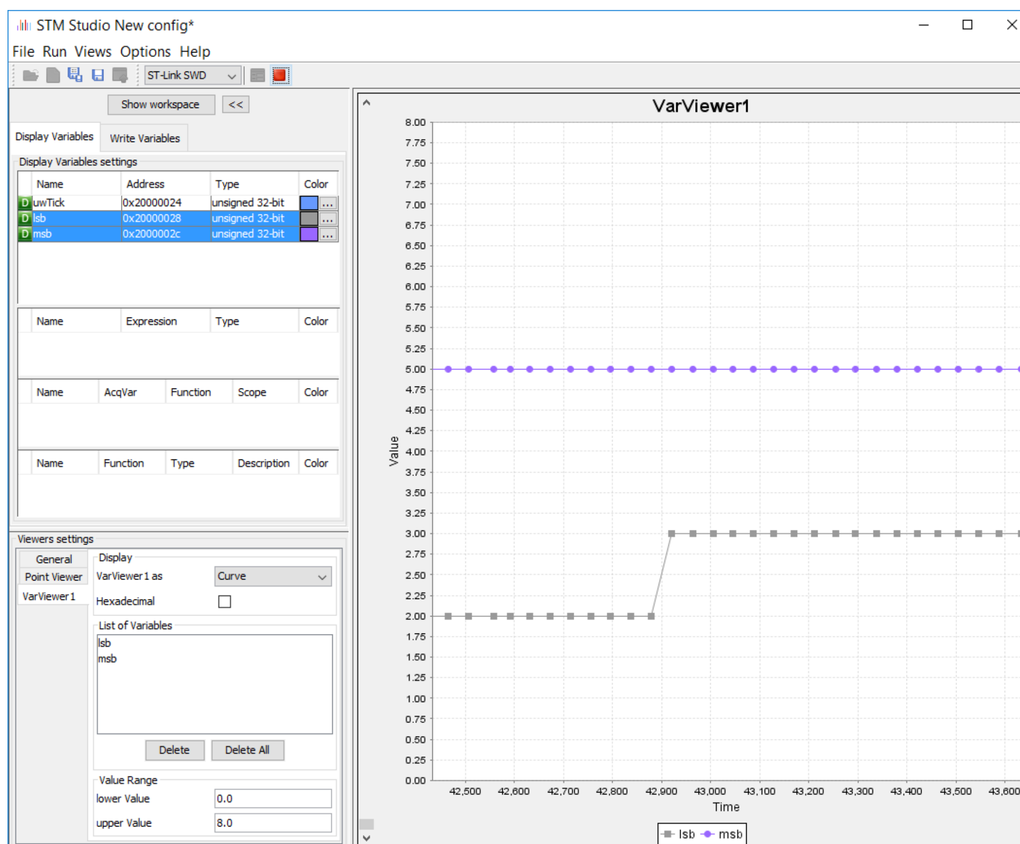
Pozostałe domyślne ustawienia aplikacji nie wymagają modyfikacji za wyjątkiem jednego. W przypadku wykresu z krzywymi domyślny przedział wyświetlanych danych jest ustawiony na jedną sekundę co w efekcie powoduje, że wyświetlane dane stosunkowo szybko się zmieniają. Można to zmodyfikować poprzez otworzenie okna właściwości wykresu (przed przystąpieniem do modyfikacji ustawień warto zatrzymać podgląd zmienny). Wystarczy na obszarze wykresu nacisnąć prawy przycisk myszy i wybrać z menu kontekstowego właściwości *Properties ...*. Po wybraniu zakładki *Plot*, a następnie zakładki *Range* (znajdującej się w dolnej części okna) możemy zmodyfikować zakres wyświetlanych danych (Rys. 36). Należy mieć na uwadze, że wprowadzone wartości podawane są w milisekundach.



Rysunek 36: Właściwości wykresu

Aby uruchomić program należy wybrać z menu opcję *Run* → *Start*. Po tej operacji zmienne będą cyklicznie odczytywane. Na Rys. 37 pokazano przykładowy wykres dla prostej aplikacji.

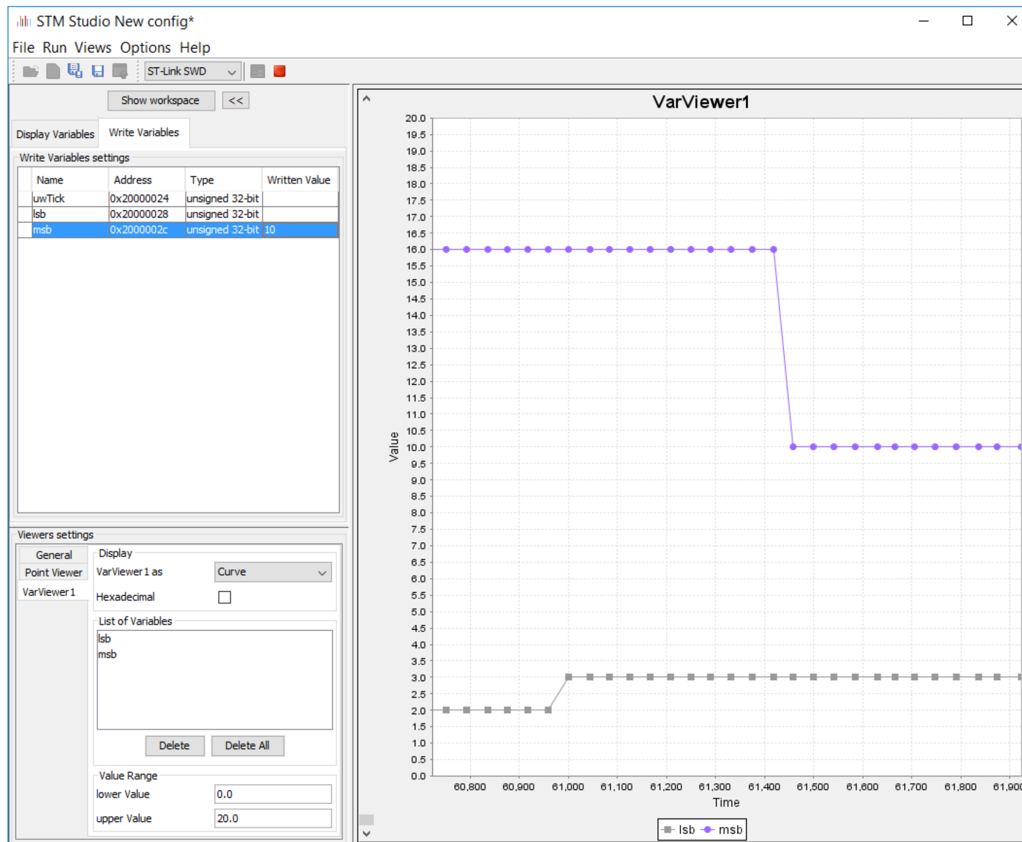
Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, edu.domski.pl



Rysunek 37: Przykładowy wykres ze zmiennymi lsb oraz msb

`STMStudio` pozwala również na modyfikowanie wartości w czasie działania programu. Zostało to pokazane na Rys. 38. Operacja ta wymaga dodania zmiennych do zakładki *Write Variables*. Można to wykonać klikając prawym przyciskiem myszy na puste pole w zakładce i wymierając pozycje importu z menu (*Import*). Pojawi się wówczas jeszcze raz okno importu (Rys. 33). Innym sposobem na dodanie zmiennych do zakładki *Write Variables* jest przeciągnięcie już istniejących wpisów z *Display Variables*. Spowoduje to ich skopiowanie do zakładki *Write Variables*. Teraz w oknie *Write Variable settings* można dokonać modyfikacji przechowywanych wartości w pamięci. Wystarczy w tym celu dwukrotnie kliknąć w komórce w kolumnie *Written Value* dla wybranej zmiennej. Po wpisaniu wartości zatwierdzamy zmianę przez wciśnięcie klawisza *Enter*.

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, `edu.domski.pl`



Rysunek 38: Modyfikacja wartości przechowywanej w zmiennej msb

8 Zadania do wykonania

8.1 Dodawanie breakpointów

Wykonaj operację dodawania *breakpointa*, na linii:

```
1 HAL_GPIO_TogglePin(LD2_GPIO_Port, LD2_Pin);
```

a następnie zaobserwuj zachowanie programu.

8.2 Poruszanie się po programie

Przećwicz poruszanie się po kodzie programu za pomocą poleceń *Resume*, *Step Into* i *Step Over*. Nie kończ wykonywania programu. Doprowadź do sytuacji, w której program zatrzyma się na inkrementacji zmiennej *lsb*. Najeżdżając kursorem na nazwę zmiennej możliwe jest podglądniecie jej aktualnej wartości (rysunek 9). Co możesz powiedzieć o tej funkcjonalności, jakie daje ona możliwości?

8.3 Podgląd zmiennych

Przećwicz dodawanie nowych wyrażeń oraz ich usuwanie (rozwinięcie menu kontekstowego i wybranie polecenia *Remove* na wybranym wyrażeniu).

8.4 Podgląd rejestrów

Co można wywnioskować z informacji przedstawionych na rysunkach 12 i 13? Szczegółowy opis rejestrów można znaleźć w [12]. Kiedy aktualizowany jest stan rejestrów?

8.5 Przekierowanie funkcji printf() z wykorzystanie interfejsu USART

Skonfiguruj projekt, tak aby interfejs USART był aktywny i wykorzystywany. Jako interfejs wyjściowy dla funkcji `printf()` można wykorzystać peryferium USART2. Należy odpowiednio skonfigurować owy

interfejs wewnątrz programu `STM32CubeMX`. Można to zrobić przez wybranie asynchronicznego trybu pracy dla USART2 (*Pinout* → *USART2* → *Mode* → *Asynchronous*) (rysunek **??**).

Co świadczy o tym, że peryferium skonfigurowane jest poprawnie?

Prędkość transmisji dla USART2 powinna być ustawiona na 115200 $\frac{bit}{s}$ (*Configuration* → *Connectivity* → *USART2* → *Parameter Settings* → *Basic Parameters* → *Baud Rate*). Ponadto, należy upewnić się, że długość słowa (ang. *Word Length*) to 8, parzystość (ang. *Parity*): brak (ang. *None*) oraz liczba bitów stopu (ang. *Stop Bits*) to 1. Pozostałe parametry należy pozostawić bez zmian.

Uwaga! Pamiętaj o ponownym wygenerowaniu kodu.

W funkcji `_write()` powinno się znaleźć następujące wywołanie:

```
1 HAL_UART_Transmit(&huart2,ptr,len,50);
```

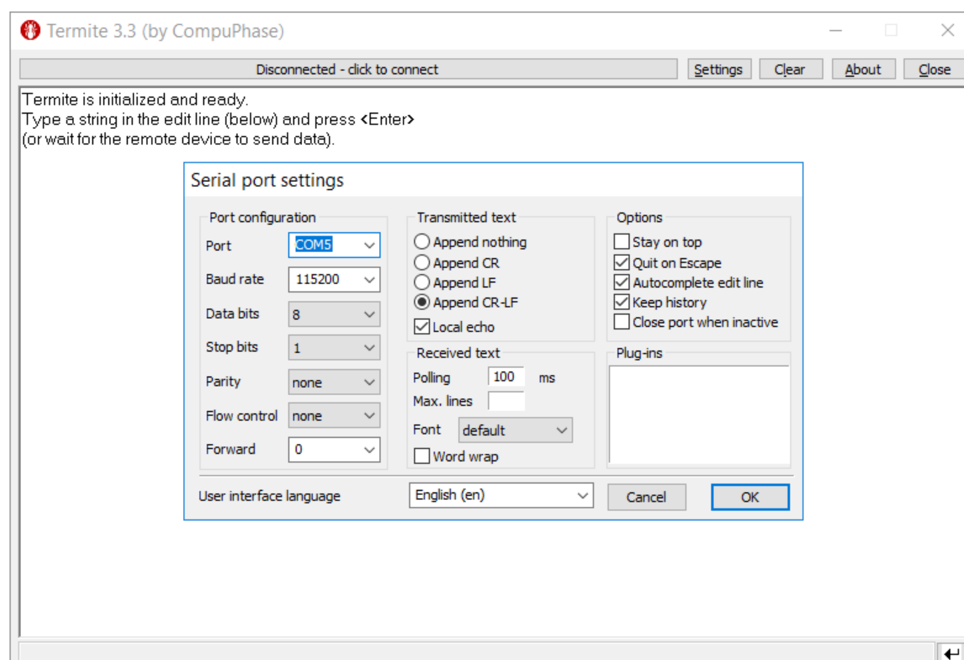
Na czym polega proces przekierowania wyjścia funkcji `printf()`? Dlaczego należy redefiniować funkcję `_write()`?

Dodaj do programu klika wywołań funkcji `printf()` i przetestuj działanie programu.

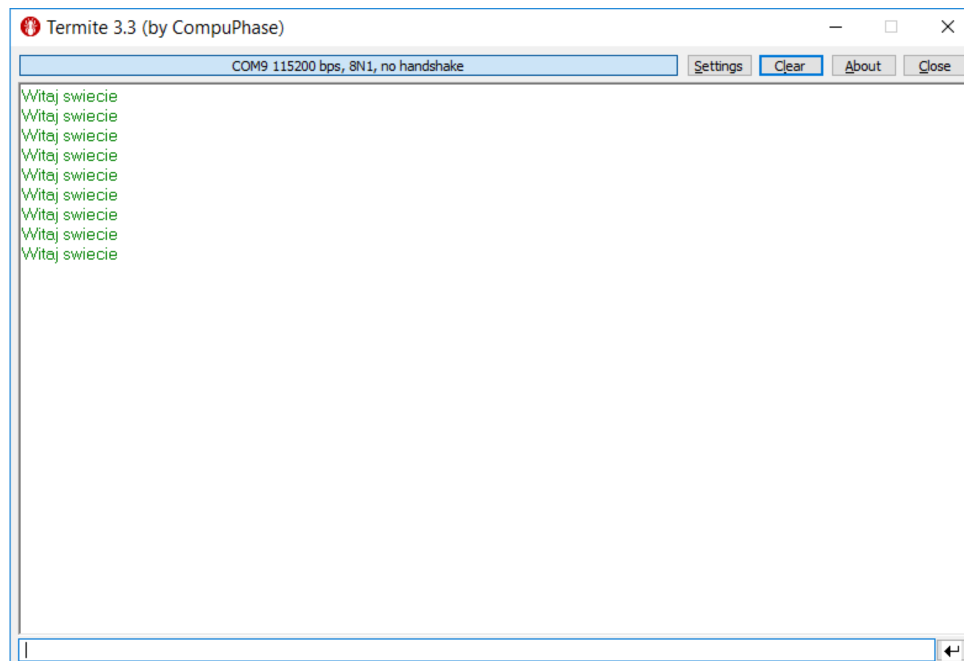
Jako terminal można wykorzystać niewielką aplikację `Termite` [4] lub `PuTTY` [13]. Preferencyjnie należy użyć aplikacji `Termite` bez instalatora (ang. *stand–alone*).

Aby zidentyfikować na jakim porcie dostępny jest emulowany port szeregowy i połączyć się z nim należy kolejno:

1. Odłączyć płytkę deweloperską od komputera.
2. Uruchomić program `Termite` oraz przejść do ustawień *Settings*, jak zostało pokazane na rysunku 39.
3. Z listy *Port configuration* → *Port* odczytać oraz zanotować jakie dostępne są porty szeregowe.
4. Zamknąć okno konfiguracji.
5. Podłączyć płytkę deweloperską do komputera.
6. Ponownie otworzyć okno ustawień *Settings* programu `Termite`.
7. Z listy *Port configuration* → *Port* wybrać port, który wcześniej nie był obecny.
8. Wybrać ustawienia zgodne z konfiguracją programu w `STM32CubeMX` (prędkość 115200 $\frac{b}{s}$, brak parzystości, 1 bit stopu).
9. Zapisać ustawienia naciskając na przycisk *Ok*, wówczas `Termite` powinien się połączyć (rys. 40).



Rysunek 39: Ustawienia programu Termite



Rysunek 40: Przykładowa transmisja z wykorzystaniem programu Termite i płytki deweloperskiej

W jaki sposób odbywa się transmisja danych, jakie urządzenia pośredniczą w transmisji [11]?

8.6 Przekierowanie funkcji printf() z wykorzystanie interfejsu SWV

Wykorzystaj interfejs SWV, aby zrealizować te same operacje co podczas realizacji zadania z wykorzystaniem interfejsu komunikacji szeregowej USART.

Uwaga! Pamiętaj o ponownym wygenerowaniu kodu.

Redefinicja funkcji `_write()` powinna wyglądać następująco:

```
int _write(int file, char *ptr, int len) {
  int i;
  for (i = 0; i < len; i++) {
    ITM_SendChar(*ptr++);
  }
  return len;
}
```

Czym jest i do czego służy SWV? Jaka jest rola pinu SWO?

8.7 STMStudio – rysowanie zawartości zmiennych na ekranie

Stwórz w `STMStudio` dwa wykresy. Pierwszy wykresem powinien być wykres słupkowy (ang. *bar graph*) ze zmiennymi *lsb* oraz *msb*. Natomiast drugi wykres powinien być typu "z krzywymi" (ang. *Curve*), gdzie powinny być umieszczone trzy zmienne: *uwTick*, *lsb* i *msb*.

8.8 Uporządkowanie stanowiska

Odłóż płytkę i kabel na miejsce. Usuń projekt z Atollic TrueSTUDIO. Można to zrobić przez kliknięcie prawym przyciskiem myszki na projekt i wybranie opcji Usuń (Delete) z menu kontekstowego.

9 Podsumowanie

W ćwiczeniu zostały przedstawione techniki związane z usuwaniem błędów z oprogramowania. Ponadto, przedstawione zostały różne sposoby pozwalające na śledzenie wykonywanego kodu, jak przekierowanie wyjścia funkcji `printf()`. Ponadto, zaprezentowano narzędzi `STMStudio` pozwalające na wizualizację danych przechowywanych w zmiennych programu, a także ich modyfikację.

Literatura

[1] Open On-Chip Debugger, Free and Open On-Chip Debugging, In-System Programming and Boundary-Scan Testing. http://openocd.org/.

[2] TrueSTUDIO - Atollic - ST. https://atollic.com/truestudio.

[3] Atollic. Cortex-M debugging: Introduction to Serial Wire Viewer (SWV) event- and data tracing. http://blog.atollic.com/cortex-m-debugging-introduction-to-serial-wire-viewer-swv-event-and-data-tracing.

[4] CompuPhase. Termite: a simple RS232 terminal. https://www.compuphase.com/software_termite.htm.

[5] W. Domski. Sterowniki robotów, Laboratorium – Wprowadzenie, Wykorzystanie narzędzi STM32CubeMX oraz SW4STM32 do budowy programu mrugającej diody z obsługą przycisku. Marzec, 2017.

[6] ST. *Getting started with STM-STUDIO, User manual.*, Październik, 2013.

[7] ST. *STM32 configuration and initialization C code generation.*, Kwiecień, 2016.

[8] ST. *ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32.*, Marzec, 2016.

[9] ST. *ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32, User manual.*, Marzec, 2016.

[10] ST. *STM Studio run-time variables monitoring and visualization tool for STM32 microcontrollers.*, Marzec, 2016.

[11] ST. *STM32 Nucleo-64 board, User manual.*, Listopad, 2016.

[12] ST. *STM32L4x5 and STM32L4x6 advanced ARM®-based 32-bit MCUs, Reference Manual.*, Marzec, 2017.

[13] PuTTY. Download PuTTY - a free SSH and telnet client for Windows. http://www.putty.org/.

Wszystkie prawa zastrzeżone Wojciech Domski © 2017-2018, edu.domski.pl