Okładka 1

Pierwsze kroki w STM32

z wykorzystaniem CubeIDE oraz HAL
nauka dobrych nawyków

Podczas tworzenia dzieła nie ucierpiał ani jeden rezystor!

Sławomir Skrzyński
sas.ze@vp.pl
Wersja z dnia 3.04.2026 godzina 11:09

Spis treści

Wstęp

Motorem napędowym stworzenia książki była chęć zainteresowania początkujących użytkowników Arduino czymś nowocześniejszym niż **przestarzałe, powolne o małych zasobach** i co ważne **drogie AVR**. Wielu użytkowników AVR czy Arduino twierdzi, że inne rozwiązania są skomplikowane i drogie. Niestety taką propagandę szerzą osoby, które nic poza AVR i Arduino nie znają. Autor książki doskonale zna zarówno AVR jak i STM32 oraz inne mikrokontrolery i mikroprocesory (6502, Z80, Z8, PIC, 8051, MC680x0) dlatego nie ma „klapek" na oczach. Przez długi czas współpracy z AVT stworzył on kilkadziesiąt projektów na AVR: projekty AVT, w tym wiele niebanalnych jak na AVR, przykładowo: Emulator & Skaner 1-Wire

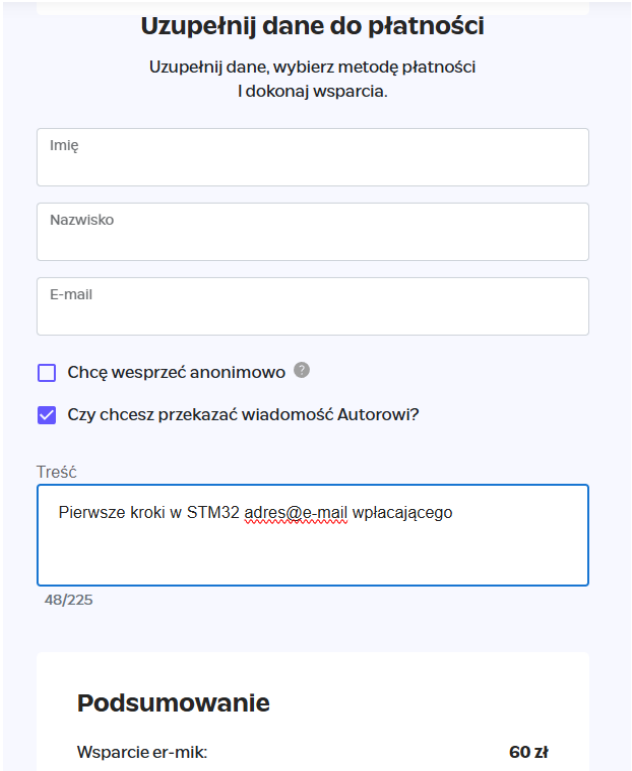
i kilkanaście na STM32. Aktualnie używa tylko STM32 co zaowocowało powstaniem książki dla zaawansowanych: Kurs programowania STM32.pdf

Chcąc wypełnić lukę pomiędzy niskiej jakości kursami w Internecie a kosztownymi dobrymi kursami STM32 napisał tanią książkę, która w przystępny sposób zachęci do używania wcale nie trudnych STM32. Kurs nie jest kursem C. Autor zakłada podstawowa znajomość C.

Przedpłaty na książkę można można dokonać wspierając autora kwotą 60zł przez serwis:

https://suppi.pl/ermik

W wiadomości dla autora należy napisać „Pierwsze kroki w STM32" oraz swój e-mail:



Osoby, które uiszczą przedpłatę otrzymają finalną wersję książki w PDF bez względu na jej końcową cenę. Ponadto będą dostawać kolejne rozdziały w trakcie ich powstawania jeszcze przed ukazaniem się finalnego produktu.

Kurs w dużej mierze będzie darmowy dlatego w początkowej fazie jego rozwoju nie będę wysyłał linków do pełnej wersji bo jej jeszcze nie będzie. Wstępnie szczuję, że darmowa wersja będzie do rozdziału 15. Można więc się zastanawiać czy warto wspierać powstanie książki wcześniej? Warto bo najprawdopodobniej cena kursu wzrośnie.

Rozdział 1. Wybór zestawu startowego

Zastanawiałem się nad wyborem pomiędzy NUCLEO-G491RE a NUCLEO-H723ZG

https://youtu.be/M6gT-tXkon8.

Po zastanowieniu wybór padł na H723 ze względu na to, że mikrokontroler (uC) ma bogate wyposażenie a płytka nie jest dużo droższa od znacznie uboższej z G491 dzięki czemu przyda się w bardziej zaawansowanych projektach i kursach.

1.1 Zalecane składniki zestawu startowego

Do kursu będziemy potrzebować:

szt.	netto	VAT 23,00%	brutto	Link
1	110,54 zł	25,42 zł	135,96 zł	https://kamami.pl/stm-nucleo-144/581677-nucleo-h723zg-zestaw-startowy-z-mikrokontrolerem-z-rodziny-stm32-stm32h723zg-5906623436453.html
1	25,60 zł	5,89 zł	31,49 zł	https://kamami.pl/moduly-rtc/575708-kamodm41t82-modul-z-zegarem-rtc-5906623432714.html
1	13,74 zł	3,16 zł	16,90 zł	https://kamami.pl/kamod-kamami/586091-modul-8-kanalowego-ekspandera-io-z-interfejsem-i2c-5906623432837.html
1	11,30 zł	2,60 zł	13,90 zł	https://kamami.pl/czujniki-temperatury/572557-kamodds18b20-modul-z-czujnikiem-temperatury-ds18b20-firmy-dallas-5906623432684.html
1	22,68 zł	5,22 zł	27,90 zł	https://kamami.pl/kamod-kamami/571786-kamodmpl3115a2-modul-z-czujnikiem-cisnienia-atmosferycznego-5906623432622.html
1	16,18 zł	3,72 zł	19,90 zł	https://kamami.pl/czujniki-temperatury/102922-kamodtem-modul-z-cyfrowym-czujnikiem-temperatury-i-termostatem-z-interfejsem-i2c-mcp9801-5906623432288.html
1	72,36 zł	16,64 zł	89,00 zł	https://kamami.pl/kamod-kamami/184717-kamodtouch-modul-6-przyciskowa-klawiatura-dotykowa-z-ukladem-at42qt1060-5906623432370.html
1	39,32 zł	9,04 zł	48,36 zł	https://kamami.pl/sterowniki-led/560872-shield-do-zestawu-maximator-5902186329621.html
1	8,86 zł	2,04 zł	10,90 zł	https://kamami.pl/kamod-kamami/558041-kamodmicelectret-modul-mikrofonu-z-wbudowanym-wzmacniaczem-5906623433247.html
1	23,97 zł	5,51 zł	29,48 zł	https://kamami.pl/audio/1188549-kamodtda2050-wzmacniacz-mocy-135-w-5906623433414.html
1	4,17 zł	0,96 zł	5,13 zł	https://kamami.pl/moduly-wzmacniaczy/1181590-modul-ze-wzmacniaczem-audio-mono-tda2030a-18w-6-12v-5906623444694.html
1	46,98 zł	10,81 zł	57,79 zł	https://allegro.pl/oferta/wyswietlacz-lcd-tft-2-4-dotykowy-spi-ili9341-obsluga-kart-sd-rysik-12022687220?bi_s=ads&bi_m=productlisting:desktop:query&bi_c=NTEzNTI2NmUtMGQ1OC00YzJjLTliYTMtYjFmNWFjZTE2OTViAA&bi_t=ape&referrer=proxy&emission_unit_id=af82e856-02be-47f6-9ced-6cc9f06b3c4c
1	60,66 zł	13,95 zł	74,61 zł	https://allegro.pl/oferta/drukarka-termiczna-em5820-usb-rs232-ttl-rolka-etykiet-18219626272
1	5,20 zł	1,20 zł	6,40 zł	Moduł z PCF8574
Suma	**14**	**461,56 zł**	**106,16 zł**	**567,72 zł**

Skład będzie ulegał zmianie aby ograniczyć koszt zestawu.

Rozdział 2. Instalacja środowiska programistycznego

Wybór padł na popularne CubeIDE w wersji 1.9.0. Dlaczego nie najnowsze w chwili pisania kursu 2.00? Ze względu na niekorzystne dla początkujących modyfikacje w oprogramowaniu.

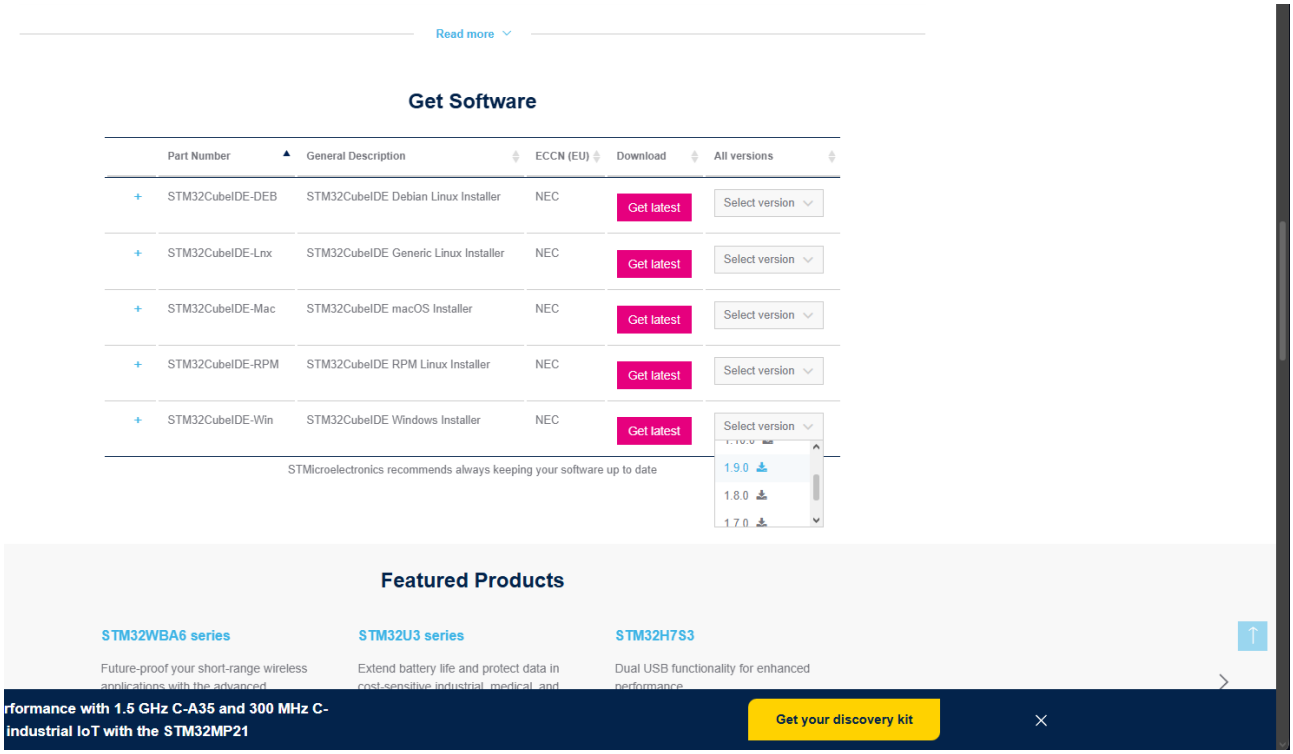
2.1 Pobranie oprogramowania

Zanim pobierzemy oprogramowanie zalecam założyć konto się na stronie producenta STM32: https://www.st.com/content/st_com/en.html

Nie jest ona konieczna ale zaoszczędzi sporo czasu na zbędne wypełnianie formularza akceptacji licencji przy pobieraniu każdego pliku (programu, manuala, noty katalogowej, itp.). Program znajdziemy w:

https://www.st.com/en/development-tools/stm32cubeide.html

Wybieramy wersję programu i system operacyjny (nie wiem czy Windows można nazywać systemem operacyjnym):



Rysunek r02-1

Następnie naciskamy „Get latest”. Jeżeli jesteśmy zalogowani pobieranie rozpocznie się zaraz po zaakceptowaniu licencji, jeżeli nie jesteśmy zalogowani, będziemy musieli wypełnić dodatkowo formularz.

2.2 Instalacja

Jest standardowa i nie wymaga omawiania.

2.3 Konfiguracja

W zasadzie nie jest wymagana. Zalecane zmiany będą omówione przy okazji pierwszego programu.

2.4 Tworzenie projektu

2.4.1 Konfigurowanie zegarów

Nie zawsze konieczne ale AVR-owcy o tym nie wiedzą i sieją propagandę! Fakt, duże możliwości powodują, że konfiguracja zegarów jest skomplikowana ale na rejestrach, po co jednak mamy CubeMX?
Temat zegarów jest rozległy dlatego wyjaśnienia znajdziecie w filmie: https://youtu.be/j3koUp-GLH0
Wewnętrzny RC jest tak stabilny, że w przeciwieństwie do **drogich** i przestarzałych AVR może taktować UART a nawet bardziej wymagające USB.
Jeden z komentarzy zmusił mnie do refleksji i przypomniało mi się, że często zegarów nie trzeba konfigurować. Dlatego nagrałem film poruszający ten temat: https://youtu.be/D9qHQXLZwno

2.4.2 Konfiguracja GPIO

2.4.2.1 Nadawanie wyprowadzeniom nazw widocznych w programie

Korzystając CubeMX nie musimy do identyfikacji wyprowadzeń używać nieczytelnych nazw GPIOx + GPIO_PIN_x ale czytelnych nazw, które sami nadajemy. Jak to zrobić? Zamiast dziesiątek rysunków i tysięcy liter, niespełna dziesięciominutowy film:
https://youtu.be/gtRg7oEnpks

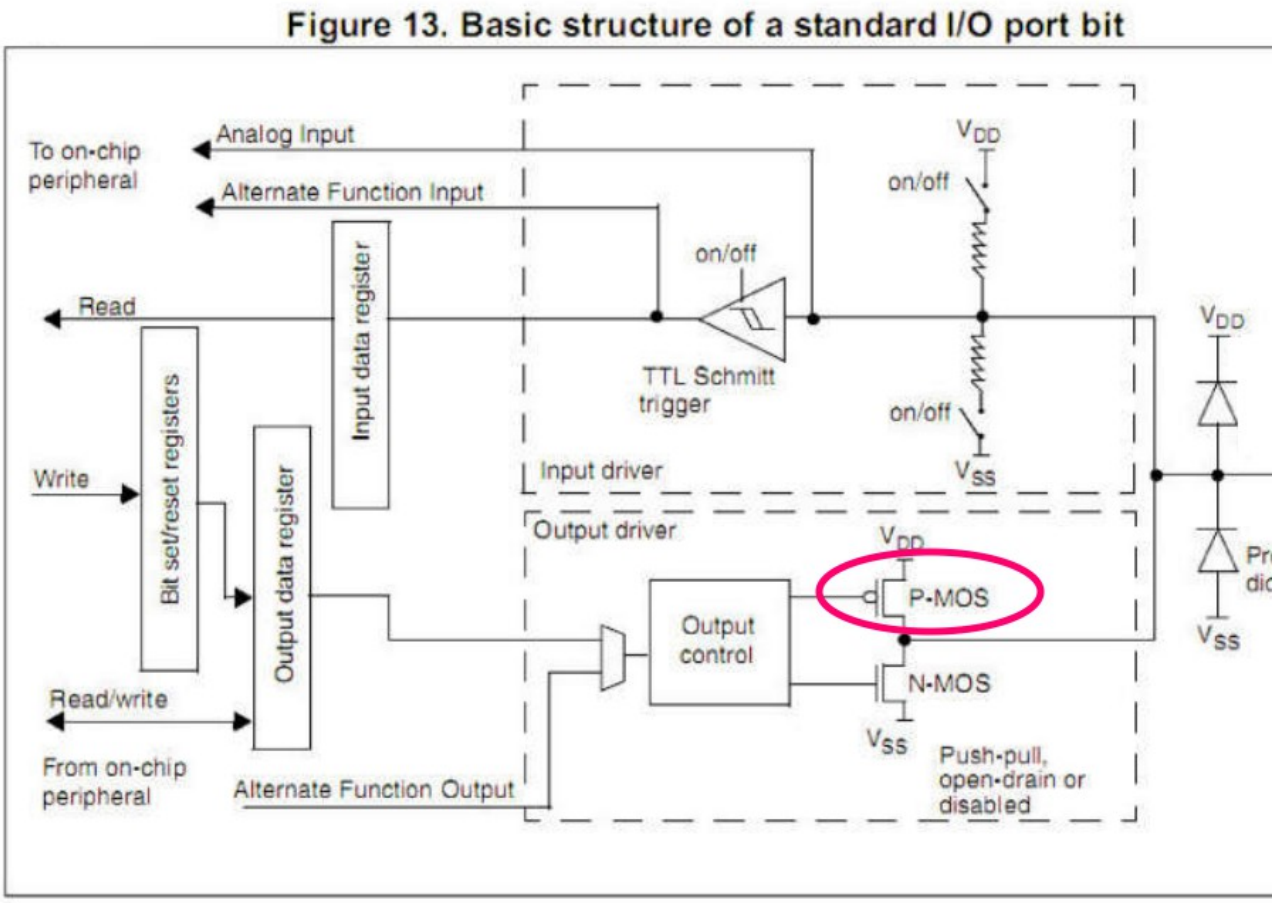
2.4.2.2 Wbudowane Podciąganie/ściąganie

Podciąganie wejść było znane już za czasów 8051 czy PCF8574. W AVR można je włączać lub nie. W STM32 dodatkowo można włączyć ściąganie wejść do masy. Mało tego podciąganie można stosować dla wyjść. Wydaje się to bez sensu ale tak nie jest w przypadku wyjść OD (rozdział 2.4.2.3). Szczegóły wyjaśniam w filmie: https://youtu.be/AMpYTq85pV8

2.4.2.3 Tryb OD (open drain – otwarty dren), „szybkość" pracy poru wyjściowego

Odpowiednik OC (otwarty kolektor) tyle, że dla układów w wykonaniu MOS. Typowe GPIO w

STM32:



Rysunek r2-2

W AVR GPIO jest zbudowane podobnie tyle, że w STM32 można wyłączyć wyjściowy tranzystor P-MOS (oznaczony czerwoną elipsą na rysunku r2-2) a w AVR nie. Jak więc w AVR uzyskuje się funkcjonalność OD? Do rejestru danych zapisujemy zero a do kierunku: 1 (porty wyjściowy) gdy chcemy ustawić port w stan zero lub 0 (port wejściowy) gdy gdy chcemy ostawić stan OD. Proste i logiczne prawda :-) W podobny sposób bramką 3-stanową symuluje się OD w układach programowalnych (CPLD/FPGA).

W STM32 wybiera się tryb pracy OD a do portu wyjściowego zapisujemy 0 gdy chcemy ustawić zero lub 1 gdy chcemy ustawić stan OD. Jak skonfigurować tryb OD pokażę w filmie:

https://youtu.be/UdcPmKRLkiM

Dodatkowo omówiłem ustawienie „szybkości" działania portu wyjściowego czyli ustawienie czasów narastania/opadania sygnału.

2.4.3 Konfiguracja interfejsów komunikacyjnych

STM32 posiada bardzo dużo interfejsów. Z tego powodu do jednego wyprowadzenia przypisane może być wiele z nich. Oczywiście w danej chwili tylko jeden z nich. W małych mikrokontrolerach

pojawia się problemu równoczesnego wykorzystania dwu lub więcej interfejsów przypisanych do tego samego wyprowadzenia. Dlatego STM umożliwił zmianę domyślnej lokalizacji interfejsu na kilka innych wyprowadzeń. W nowych STM32 zwykle są to trzy dodatkowe lokalizacje w starych tylko jedna. Ponadto w starych STM przenosimy wszystkie linie interfejsu przykładowo dla I2C DSA i SCL, dla SPI SCK, MOSI i MISO. W nowszych możemy przenieść tylko jedną z linii, przykładowo tylko SCK. inne pozostawiając bez zmian albo przenieść w inne alternatywne miejsce. Łatwiej będzie to zrozumieć oglądając filmy w dalszej części materiału.

2.4.3.1 Funkcje alternatywne GPIO

Wiele GPIO posiada funkcje alternatywne jak PWM, UART, USB, itp. Gdy jedno wyprowadzenie pełni funkcje kilku interfejsów, wybrać możemy tylko jeden z nich. Aby skorzystać z kolejnego interfejsu trzeba przenieść jego funkcjonalność na inne wyprowadzenia. Jak to zrobić wyjaśniam w filmie: https://youtu.be/ObRZ24rveC4

2.4.3.2 Wbudowane podciąganie wejść

Zwykle jest niewystarczające, przykładowo I2C wymaga rezystorów nie większych niż 5k a wbudowane ma około 40 (od 30 do 50). Takie podciąganie nie jest wystarczające do normalnej pracy interfejsu, chyba, że to interfejs przycisku mechanicznego ale można ustawić stan spoczynkowy lub wręcz przeciwnie (awarii) interfejsu.
Opcja 1 powoduje, że interfejs „nie widzi" danych gdy jest coś nie tak (np. brak połączenia).
Opcja 2 w takiej sytuacji da stan alarmowy (dla UART sygnał BREAK, dla I2C przegranie arbitrażu mimo że na magistrali jest tylko jeden master). Wyjaśnienie w filmie: https://youtu.be/L9irLkgWIDM

2.4.3.3 Wyłączenie części pinów interfejsu

Gdy nie potrzebujemy wszystkich pinów interfejsu przykładowo SPI tylko wysyła do slave możemy zaoszczędzić nieużywane linie w tym przypadku MISO przeznaczając ten pin na inne cele. Opcja bardzo przydatna w STM32 o małej liczbie wyprowadzeń. RX w UART gdy tylko nadajemy. Tego AVR nie mają. Jak to działa zobaczysz w filmie: https://youtu.be/NqMdCHk2qWE

Rozdział 3. Pierwszy program

Pierwszym programem na komputery jest zwykle słynne „Hello Word”. Odpowiednikiem dla uC jest zamiganie LED'em.

3.1 Blink z delay. Wgrywanie softu przez DFU.

Pierwszy program:

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

        HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
        HAL_Delay(100);

  }
  /* USER CODE END 3 */
}
```

Film z prezentacją kodu: https://youtu.be/GhUgu71tAgw

Kod programu (projekt w CubeIDE): L03_01

3.2 Importowanie projektu

W filmie pokażę jak importować projekt przykładowo poprzedniej lekcji pobrany z Internetu L03_01. Pisać można by dożo, nawet bardzo dużo więc zapraszam do obejrzenia kilkuminutowego filmu: https://youtu.be/KxHNmr3_tEE

3.3 Wywalamy delay. Od teraz piszemy tylko funkcje nieblokujące

Funkcje nieblokujące „uchronią” nas przed używaniem RTOS, który rozwiązuje niektóre problemy ale generuje wiele innych. Funkcje nieblokujące pozwolą lepiej wykorzystać możliwości RTOS jak i zrozumieć mechanizmy komunikacji między zadaniami.

Prosty przykład migania diodą ma ta wadę, że delay blokuje CPU na długi czas. Pewnie zapytacie, i jaki to problem? Spróbujcie zamigać kilkoma LED'ami, każdą z inną częstotliwością. Konstrukcja:

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
```

```
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */


      HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);
      HAL_Delay(100);

      HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
      HAL_Delay(500);

      HAL_GPIO_TogglePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin);
      HAL_Delay(1000);

}
/* USER CODE END 3 */
```

nie da spodziewanych rezultatów https://youtu.be/gi3KXzEDAxQ. Jak sobie z tym poradzić? Nie trzeba sięgać po RTOS, wystarczy skorzystać z zegara systemowego.

3.4 Odpowiednik millis z Arduino

```
    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)    {
          __WFI();

          /* USER CODE END WHILE */

          /* USER CODE BEGIN 3 */

          uint32_t static tim1, tim2, tim3;

          uint32_t tim = HAL_GetTick();
          if ( tim1 < tim ){
                tim1 = tim + 100;
                HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);
          }


          if ( tim2 < tim ){
                tim2 = tim + 500;
                HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
          }


          if ( tim3 < tim ){
                tim3 = tim + 1000;
                HAL_GPIO_TogglePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin);
          }

    }
```

```
      /* USER CODE END 3 */
```

Kod: L03-2
Film: https://youtu.be/EcPv9BYxsQ8

3.5 Programowe timery

Dlaczego lepsze od poprzedniego rozwiązania? Bo zamiast czterech bajtów na licznik zazwyczaj wystarczają dwa czasem nawet jeden. Ma to duże znaczenie zwłaszcza w AVR, w których jest mało RAM a CPU jest wolny w dodatku 8-bit przez co operacje na liczbach większych niż 8-bit musi rozkładać na kilka 8-bit. Rozwiązanie te sprawdzi się też w STM32 o małych zasobach (rodzina STM32C0, G0). Skorzystamy z programowych timerów:

```
/* Private user code
---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

uint8_t volatile tim1;
uint16_t volatile tim2, tim3;
void HAL_IncTick(void){
  uwTick += (uint32_t)uwTickFreq;

  if ( tim1 ) tim1--;
  if ( tim2 ) tim2--;
  if ( tim3 ) tim3--;
}


/* USER CODE END 0 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)  {
        __WFI();

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

            if ( !tim1 ){
                  tim1 = 200;
                  HAL_GPIO_TogglePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin);
            }


            if ( !tim2 ){
                  tim2 = 1000;
                  HAL_GPIO_TogglePin(LED_RED_GPIO_Port, LED_RED_Pin);
```

```
            }


            if ( !tim3 ){
                  tim3 = 500;
                  HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);
            }


  }
  /* USER CODE END 3 */
```

Kod projektu: L03-4

Film z wyjaśnieniami: https://youtu.be/_kVRswo5hRo

W lekcjach pominąłem obsługę LED w przerwaniach. Ten temat oraz obsługa przez PWM i DMA jest poruszony w książce dla zaawansowanych.

3.6 Dodajemy WatchDoga. Niezbędny w poprawnie napisanych programach

O watchdog'u (WDG) zapominają amatorzy ale także i zaawansowani programiści. Bywa, że obsługę WDG dodają pod koniec pisania oprogramowania i przez co często nie przetestują wszystkich funkcji programu co powoduje zadziałanie WDG choć nie powinien. Dlatego warto nabrać nawyku włączania WDG w fazie tworzenia projektu. Wtedy ewentualne błędy będą wychwycone w trakcie pisania programu a nie u użytkownika. Niektórzy traktują użytkowników jako bezpłatnych testerów! Program testowy:

```
/* Private user code
---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

uint8_t volatile tim1;
uint16_t volatile timLedRead=1000, tim3;
void HAL_IncTick(void){
  uwTick += (uint32_t)uwTickFreq;

  if ( tim1 ) tim1--;
  if ( timLedRead ) timLedRead--;
  if ( tim3 ) tim3--;
}


/* USER CODE END 0 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)  {
        __WFI();
        HAL_IWDG_Refresh(&hiwdg1);

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */


            if ( !timLedRead ){
                  HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, GPIO_PIN_RESET);
            }


            if ( !tim1 ){
                  tim1 = 100;
                  HAL_GPIO_TogglePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin);
            }


            if ( !tim3 ){
                  tim3 = 500;
                  HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);
            }


            //----- sprawdzenie stanu przycisku -----//
            if ( HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) ){
                  // Nie likwidujemy drżenia styków do:
                  // 1. NUCLEO ma obwód RC,
                  // 2. do tak prostej prezentacji nie jest to potrzebne

                  HAL_Delay(600);     // Nie odświerzamy WDG

            }

  }
  /* USER CODE END 3 */
```

Kod: L03-5
Film: https://youtu.be/Ri2C_KZJm9s

Niewielka modyfikacja projektu:

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)  {
        __WFI();
        HAL_IWDG_Refresh(&hiwdg1);

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
```

```
                if ( !timLedRead ){
                        HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, GPIO_PIN_RESET);
                }


//              if ( !tim1 ){
//                      tim1 = 100;
//                      HAL_GPIO_TogglePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin);
//              }


                if ( !tim3 ){
                        tim3 = 500;
                        HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin);
                }


                //----- sprawdzenie stanu przycisku -----//
                if ( HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) && !timLedRead ){
                        HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_SET);
                        // Nie likwidujemy drżenia styków do:
                        // 1. NUCLEO ma obwód RC,
                        // 2. do tak prostej prezentacji nie jest to potrzebne

                        HAL_Delay(500); // Nie odświerzamy WDG

                }

  }
  /* USER CODE END 3 */
```

Film omawiający zmiany: https://youtu.be/QJPEkwcRSJ4
Zmieniony projekt: L03-5b

Rozdział 4. Debuger

Doskonałe narzędzie pomijane przez początkujących. Pewnie ze strachu wywołanego mitami na jego temat. Pokaże jednak, że jest bardzo pomocne i łatwe w użyciu. Bardzo przyspiesza diagnozowanie błędów i uruchamianie prototypów.

4.1 Praca krokowa

Aby uruchomić debuger naciskamy F11. Program zostanie skompilowany i zatrzyma się na pierwszej instrukcji. F8 uruchamia CPU do kolejnego punktu BREAK, jeżeli go nie ma działa bez przerwy. CPU można zatrzymać w dowolnym momencie naciskając symbol PAUSE). Pracę krokową realizujemy klawiszami F5 i F6. F6 przechodzi do kolejnej linii kodu, F5 wchodzi w funkcję jeżeli ją napotka (F6 wykona kod funkcji p oczy zatrzyma się w kolejnej linii. Szkoda pisać bo tysiące słów zastąpi jeden film: https://youtu.be/LKS5WOLsDD8

4.2 Wyjątki

W przeciwieństwie od AVR czy 8051 ARM obsługuje wyjątki jak MC680x0. Takimi wyjątkiem może być nielegalny dostęp do pamięci (przykładowo nieistniejący obszar)????? może wytłumaczyc jak to działa w 68k, czy dzielenie przez zero.

4.2.1 Dzielenie przez zero, okno analizatora błędów

Przysłowie mówi „Pamiętaj cholero nie dziel przez zero”. Cpu w takiej sytuacji generuje wyjątek co można zobaczyć w filmie: https://youtu.be/7RbBI-waahE

4.2.2 modyfikacja zmiennych

W poprzedniej lekcji pokazałem taką możliwość teraz bardziej obrazowa demonstracja: https://youtu.be/ZaRMBebTue0

4.2.3 manipulowanie peryferiami

Przykład oparłem o GPIO a konkretnie sterowanie LED'ami na płytce NUCLEO: https://youtu.be/LsT6xTOjI0A

Tak samo można modyfikować inne peryferia, jak timery (podzielnik, PWM), UART (szybkość transmisji, format ramki)

4.2.4 podgląd zmiennych on-line

Bardzo przydatna funkcją jest podgląd zmiennych na żywo. AVR o takiej możliwości może tylko pomarzyć: https://youtu.be/i6QW0mLDWjY

Rozdział 5. GPIO

5.1 Output - sterowanie LED na przerwaniach systemowych

W lekcji zastosujemy funkcje ustawiania, kasowania i przełączanie portu.
Do ustawiania, zerowania służy funkcja:
`HAL_GPIO_WritePin`

Do zmiany stanu:
`HAL_GPIO_TogglePin`

Przerwanie systemowe:
`void HAL_IncTick(void)`

Kod programu:

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */


void HAL_IncTick(void){
        uwTick += (uint32_t)uwTickFreq;


        uint16_t static timYellow = 1000, timRed = 1000, timGreen = 1000;;

        if( --timYellow == 900 ) HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_RESET );
        if( !timYellow ){
                timYellow = 1000;
                HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_SET );
        }


        if ( --timRed == 700 ) HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, GPIO_PIN_SET );
        if ( !timRed ){
                timRed = 1000;
                HAL_GPIO_WritePin(LED_RED_GPIO_Port, LED_RED_Pin, GPIO_PIN_RESET );
        }

        if ( --timGreen == 500 ) HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin );
        if ( !timGreen ) timGreen = 1000;
}


/* USER CODE END 0 */
```

Film z wyjaśnieniami: https://youtu.be/nCoR9E8vjhg
Projekt programu lekcji z filmu: L05-1

5.2 Input – odczyt stanu przycisku

Kod programu:

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

void HAL_IncTick(void){
```

```
	uwTick += (uint32_t)uwTickFreq;


	uint16_t static timGreen = 1000;;

	if( HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) )
HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_SET );
	else HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_RESET );

	if ( --timGreen == 500 ) HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin );
	if ( !timGreen ) timGreen = 1000;
}


/* USER CODE END 0 */
```

Film z wyjaśnieniami: https://youtu.be/6tyta2oqkCM

Projekt: L05-2

5.3 Likwidacja drżenia styków

Procedura pokazana poniżej:

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

#define SW_TIM      30


void HAL_IncTick(void){
	uwTick += (uint32_t)uwTickFreq;


	uint16_t static timGreen = 1000;
	int8_t static countB1, flB1;

	if( HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) ) {
		if ( countB1 < SW_TIM/2 ) countB1++; else flB1 = 1;
	}else {
		if ( countB1 > -SW_TIM/2 ) countB1--; else flB1 = 0;
	}


	if( flB1) HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_SET );
	else HAL_GPIO_WritePin(LED_YELLOW_GPIO_Port, LED_YELLOW_Pin, GPIO_PIN_RESET );


	if ( --timGreen == 500 ) HAL_GPIO_TogglePin(LED_GREEN_GPIO_Port, LED_GREEN_Pin );
	if ( !timGreen ) timGreen = 1000;
}
```

```
/* USER CODE END 0 */
```

Film z wyjaśnieniami: https://youtu.be/xRCDhHSWd8g

Projekt: L05-3

5.4 sterowanie multipleksowe wyświetlacza LED 7-segmentów

W przykładzie wykorzystana jest płytka Maximator ale można na podstawie jej dokumentacji Podłączyć prawie dowolne wyświetlacze.

Stoper z międzyczasami

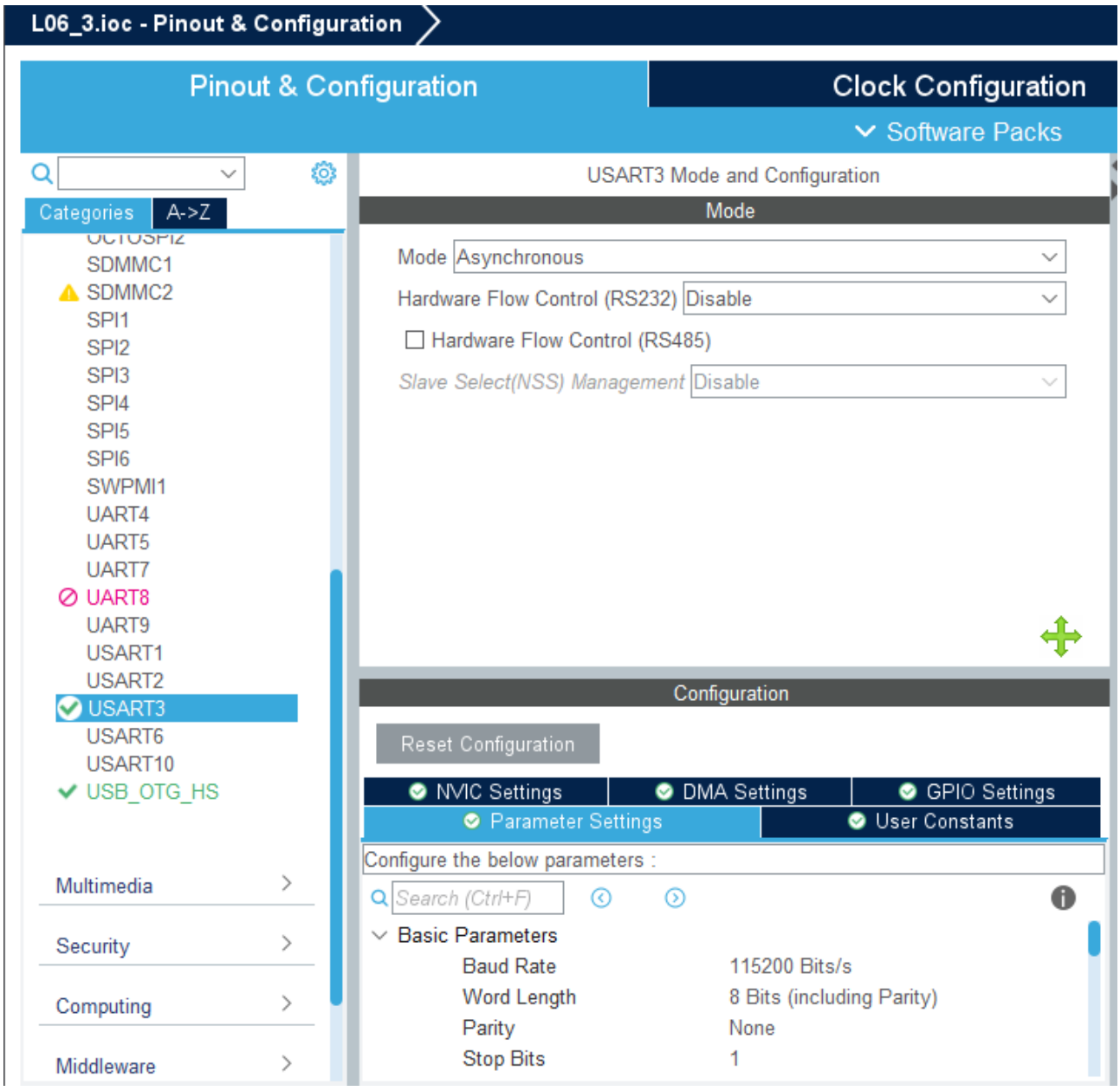
?????

Rozdział 6. UART

UART jest chyba najpopularniejszym intrefejsem komunikacyjnym w mikrokontrolerach. STM32 mają od jednego do nawet 13 UART. W razie potrzeby można zwiększyć ich liczbę układami zewnętrznymi komunikującymi się przez i2C lub SPI: [Dodatkowe porty UART w Arduino]

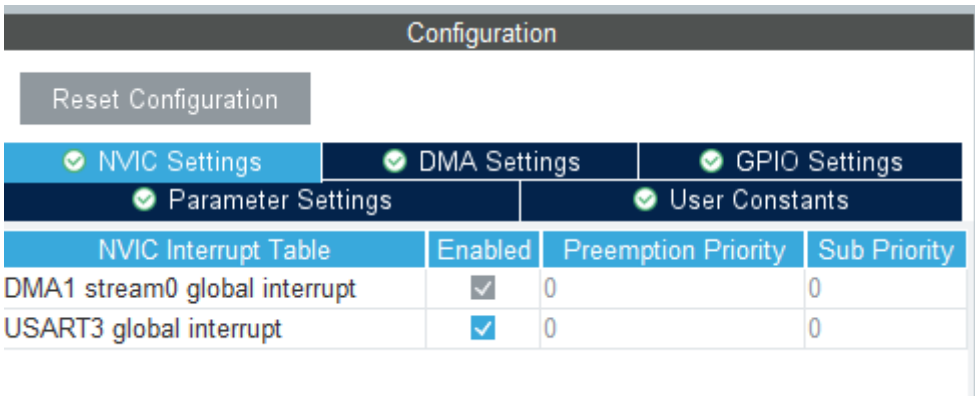
6.1 Konfiguracja UART

Włączenie (mode: Asynchronus), ustawienie prędkości i formatu ramki 115200 8N1:



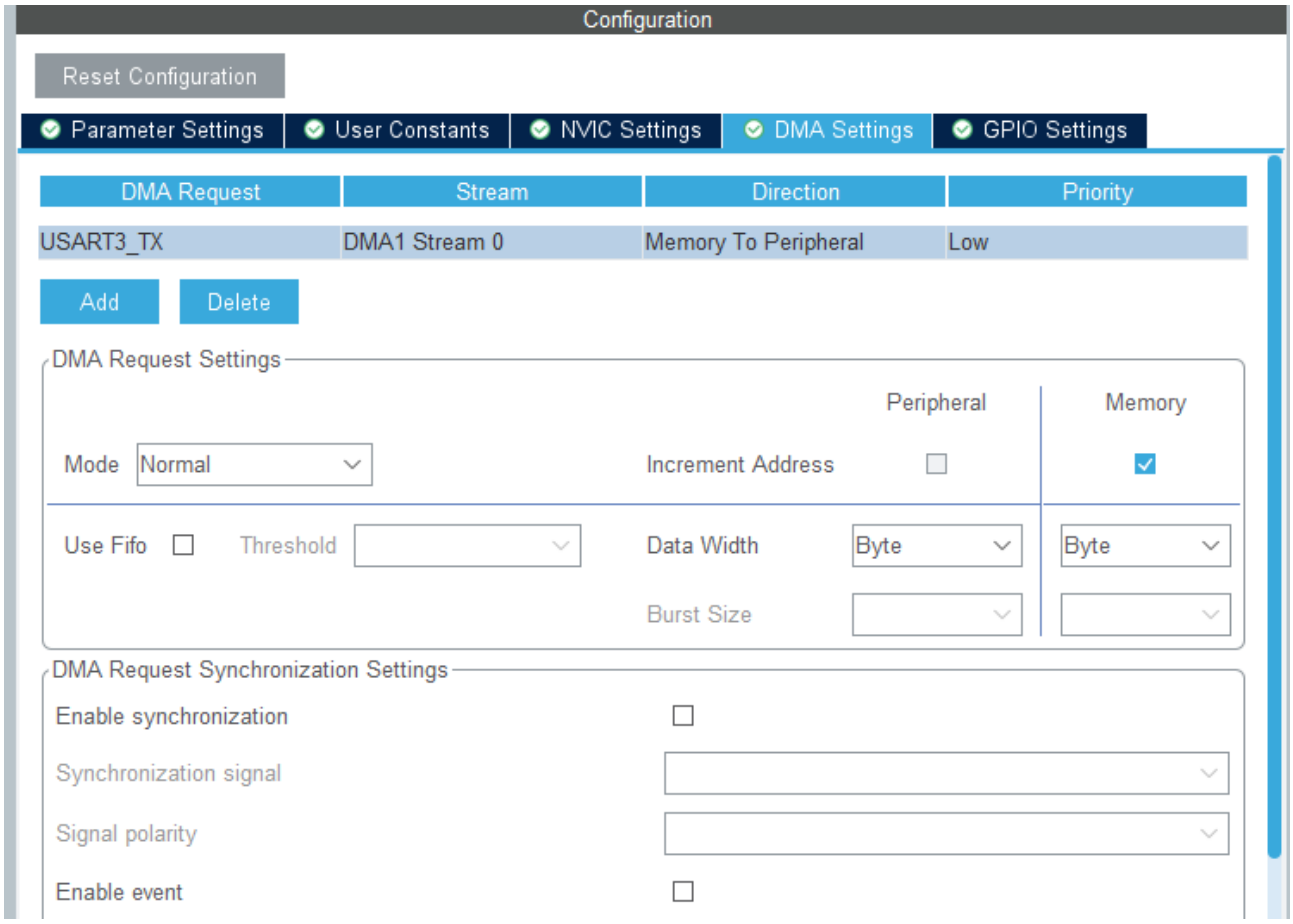
Rysunek r06-1_0

Włączenie przerwań:



Rysunek r06-1_1

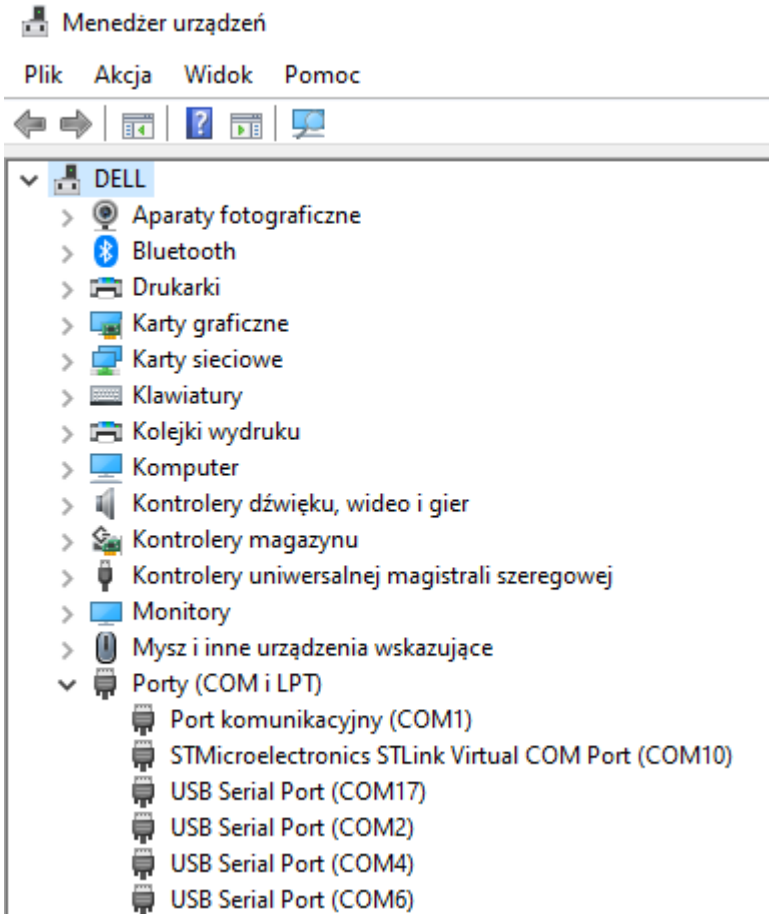
oraz DMA:



Rysunek r06-1_2

6.2 Terminal, wybranie VCOM.

Abu poznać numer portu COM płytki NUCCLEO, w menadżerze urządzeń (Win+x) szukamy portu COM o nazwie „STLink Virtual COM”:



Rysunek r06-2

W przykładzie jest to COM numer 10.

6.3 Wysyłanie danych na przerwaniach

Pomijam wysyłanie funkcją blokującą i od razu pokaże jak to robić na przerwaniach. Jeżeli komuś zależy na funkcji blokującej to wystarczy usunąć sufiks „_IT” w nazwie funkcji o dodać jeden parametr (timeout). Co ważne timeout musi być na tytle długi aby UART zdążył wysłać wszystkie znaki. Przykładowo przy 9600 8N1 czas transmisji jednego bajtu to około 1ms co wynika z wzoru 1/(9600/10) gdzie 10 to 8 bitów danych + bit startu i stopu. Aby więc wysłać tekst „***** ***” timeout musi wynosić co najmniej 8ms.

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)  {
        HAL_IWDG_Refresh(&hiwdg1);
        __WFI();

    /* USER CODE END WHILE */
```

```
    /* USER CODE BEGIN 3 */

        if ( !timUART ){
              timUART = 2000;

              const char txt[] = "Test UARTa  ";
              HAL_UART_Transmit_IT(&huart3, (uint8_t*)txt, strlen(txt) );
        }
  }
  /* USER CODE END 3 */
```

Film z wyjaśnieniami: https://youtu.be/CpGTMV6wCDs

Projekt: L06-3

6.4 Wysyłanie z użyciem DMA

Aby wysyłać używając DMA wystarczy kosmetyczna zmiana kodu:

```
    /* USER CODE BEGIN 3 */

        if ( !timUART ){
              timUART = 2000;

              const char txt[] = "Test UARTa  ";
              HAL_UART_Transmit_DMA(&huart3, (uint8_t*)txt, strlen(txt) );
        }
  }
  /* USER CODE END 3 */
```

6.5 Wysyłanie wielu komunikatów (przerwanie od zakończenia transferu)

Jeżeli chcemy wysłać kilka komunikatów po sobie nie można wywołać kilku funkcji transferu danych bo zadziała tylko pierwsze wywołanie. Aby rozwiązać ten problem należy użyć poniższego kodu:

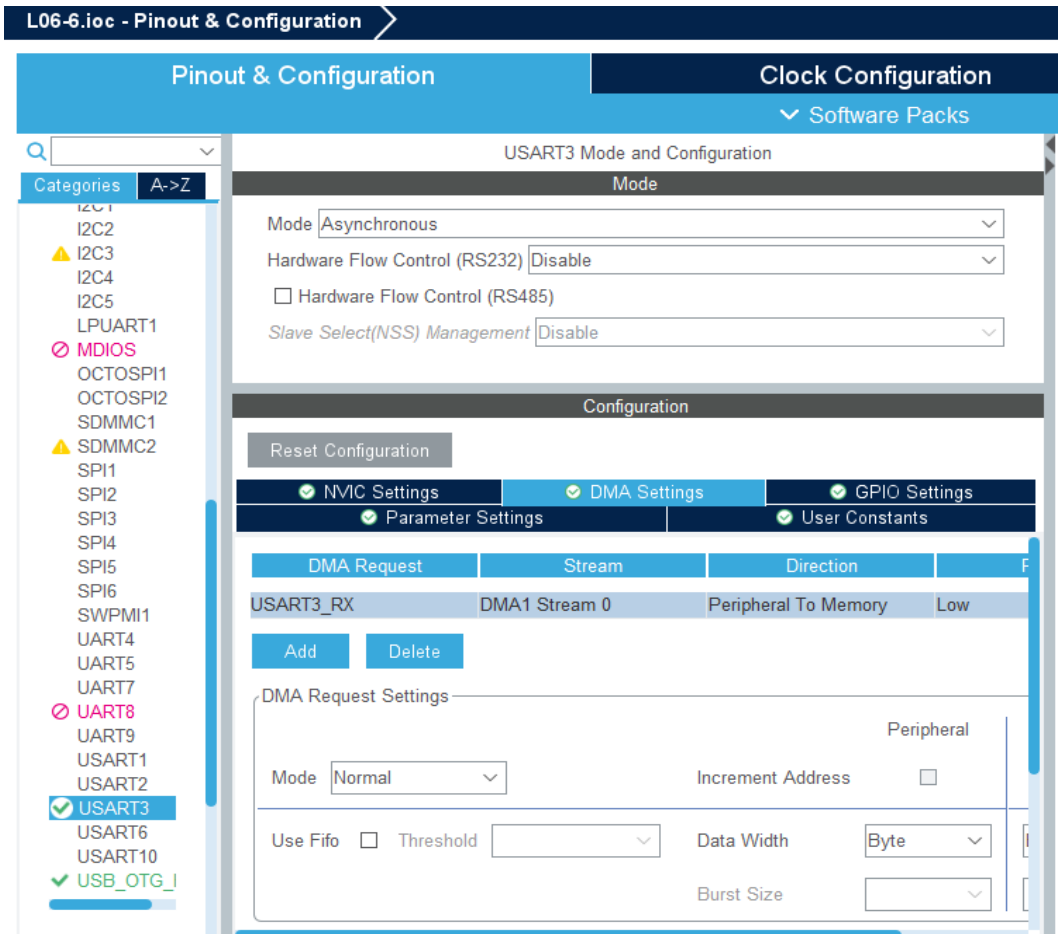
```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

uint16_t volatile timUART;
void HAL_IncTick(void){
      uwTick += (uint32_t)uwTickFreq;

      if ( timUART ) timUART--;

}


uint8_t cntComUart, lenUart;
#define      MAX_LEN_COM        50
```

```
char txtUart[5][MAX_LEN_COM];
HAL_StatusTypeDef printUART(char * txt){
        return HAL_UART_Transmit_DMA(&huart3, (uint8_t*)txt, strlen(txt) );
}


void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart){
      //jeżeli więcej UART to sprawdzaj huart
      if( --lenUart )    printUART(txtUart[cntComUart++]);
      __NOP();
}


/* USER CODE END 0 */

    /* USER CODE BEGIN 3 */

        if ( !timUART ){
              timUART = 2000;

              cntComUart = 0;
              strncpy(txtUart[0], "Test 2 ", MAX_LEN_COM);
              strncpy(txtUart[1], "Test 3 ", MAX_LEN_COM);
              lenUart = 3;
              printUART("\n\rTest 1 ");
        }


  }
  /* USER CODE END 3 */
```

Wyjaśnienie kodu w filmie: https://youtu.be/y0kYjog8Jm0
Kod: L06-5

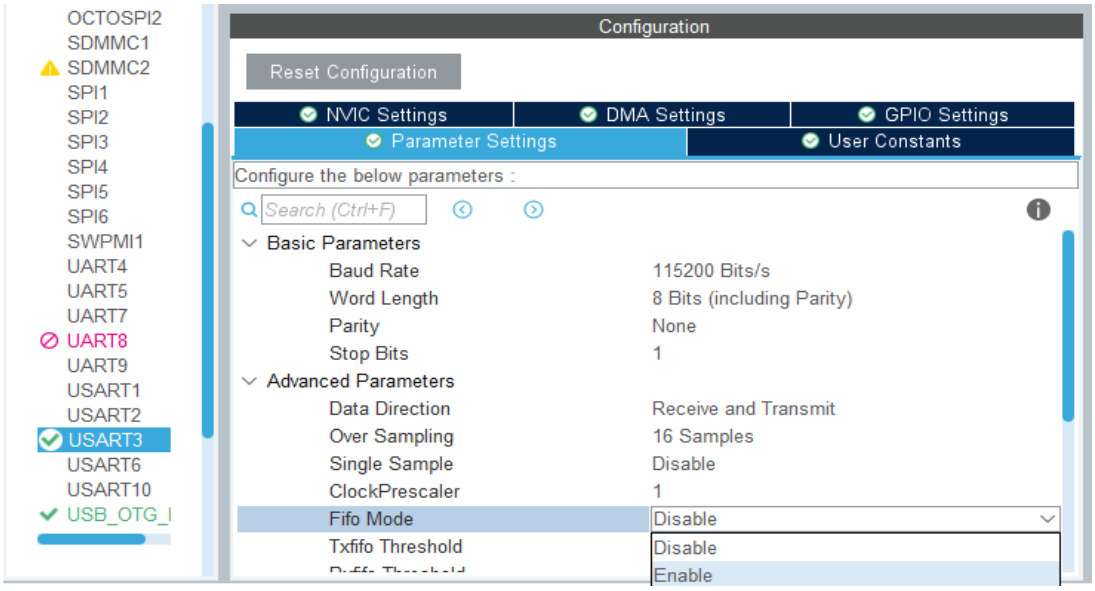
6.6 Odbiór na przerwaniach (bufor kołowy)

Przedstawione rozwiązanie będzie skuteczne przy 115200 ale większe prędkości mogą stanowić problem dlatego w przykładzie użyjemy 115200. Włączamy DMA odbiorcze, będzie potrzebne w kolejnych lekcjach:



Rysunek r06-6_1

Włączamy też FIFO jeśli jest:



Rysunek 06-6_2

FIFO jest pomocne przy większych prędkościach transmisji gdy używamy przerwań do odbioru danych. W przypadku DMA FIFO nie ma zwykle znaczenia.

Program odbioru danych, Ze względu na to, że objętość kodu jest za duża na publikację w książce, pokazuję tylko kluczowe funkcje:

```
#define      UART_RX_LEN         32
uint8_t UartRxBuf[UART_RX_LEN], UartPtrRd ,UartPtrWR, UartRxBufByte;

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){

	UartRxBuf[UartPtrWR]= UartRxBufByte;
	if ( ++UartPtrWR >= UART_RX_LEN ) UartPtrWR = 0;
	__NOP();
	HAL_UART_Receive_IT(&huart3, &UartRxBufByte, 1);
}


int16_t UartGet(){

	if ( UartPtrWR != UartPtrRd ){
		char data = UartRxBuf[UartPtrRd];
		if ( ++UartPtrRd >= UART_RX_LEN ) UartPtrRd = 0;
		timUART  = 1000;
		return data;
	}

	return -1;
}


/* USER CODE END 0 */

	/* USER CODE BEGIN 2 */

	HAL_UART_Receive_IT(&huart3, &UartRxBufByte, 1);

	/* USER CODE END 2 */

		int16_t znak;
		if ( (znak = UartGet()) != -1 ) znaki[ptr++] = znak;
```

Filmy z wyjaśnieniami:

https://youtu.be/Xj9a2v7S1Bk

https://youtu.be/bW5h9_YdRk0

Kod: L06-6

HAL nie jest wydajny i przy większych prędkościach transmisji będzie gubił dane. FIFO może poprawić sytuację ale nie każdy STM32 je posiada. Użycie szybszego CPU zwykle rozwiąże problem ale nie tędy droga. Nawet wolniejszy CPU poradzi sobie z zadaniem gdy użyjemy

programowania „na rejestrach” lub bibliotek LL co opisano w książce Kurs programowania STM32.pdf. Ale nawet w HAL zwykle można sobie poradzić o czym w kolejnych rozdziałach.

6.7 Odbiór przez DMA (nie ma sensu dla pojedynczych bajtów)

Konfiguracja UART'a jest identyczna jak do poprzedniej lekcji. Kod programu:

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

HAL_StatusTypeDef printUART(char * txt){
	return HAL_UART_Transmit_DMA(&huart3, (uint8_t*)txt, strlen(txt) );
}


uint8_t UartRxBufByte[9];
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){


	printUART( (char*)UartRxBufByte);

	HAL_UART_Receive_DMA(&huart3, UartRxBufByte, 8);
}


/* USER CODE END 0 */
```

```
  /* USER CODE BEGIN 2 */

	HAL_UART_Receive_DMA(&huart3, UartRxBufByte, 8);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)  {
			HAL_IWDG_Refresh(&hiwdg1);
			__WFI();

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
```

Film z wyjaśnieniami: https://youtu.be/KMKA4Xmcn9U

Projekt: L06-7

Po obejrzeniu filmu, można wywnioskować, że DMA nie ma większego sensu. Tak jednak nie jest bo w połączeniu z IDLE (rozdział 6.9) pozwala odbierać dane o dowolnej długości praktycznie bez angażowania do tego celu CPU.

6.8 Obsługa błędów transmisji

Temat pomijany w prawie wszystkich książkach, kursach i poradnikach bez względu czy darmowych czy płatnych. W tej kursie poradę macie za darmo!!! Specyfika HAL dla STM32 jest taka, że gdy wystąpi błąd transmisji wyłącza on dalszy odbiór danych. Aby STM32 nie „ogłuchł" w takiej sytuacji musimy obsłużyć błędy. Błędem może być zakłócenie, sygnał BREAK, czy inna przyczyna. W przypadku poprzedniej lekcji należy dodać":

```
void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart){
	HAL_UART_Receive_DMA(&huart3, UartRxBufByte, 8);
}


/* USER CODE END 0 */
```

Omówienie kodu: https://youtu.be/uUDa9u0glgI

6.9 Przerwanie IDLE od UART

Niezwykle przydatne przerwanie. Bardzo wydajne zwłaszcza z DMA dlatego przykład będzie z jego wykorzystaniem. Oczywiście można używać z odbiorem na przerwaniach, wystarczy zamienić sufiks „_DMA" na „_IT" godząc się na mniejszą wydajność. Program:

```
uint8_t UartBufRx[2000];
void HAL_UARTEx_RxEventCallback(UART_HandleTypeDef *huart, uint16_t Size){

	printUART( (char*)UartBufRx	HAL_UARTEx_ReceiveToIdle_DMA(&huart3, UartBufRx, 2000);
	HAL_UARTEx_ReceiveToIdle_DMA(&huart3, UartBufRx, 2000);
	__NOP();
}

/* USER CODE END 0 */

	/* USER CODE BEGIN 2 */

	HAL_UARTEx_ReceiveToIdle_DMA(&huart3, UartBufRx, 2000);

	/* USER CODE END 2 */
```

Film z wyjaśnieniami: https://youtu.be/SqzZI_n36e8

Kod do pobrania: L06-9

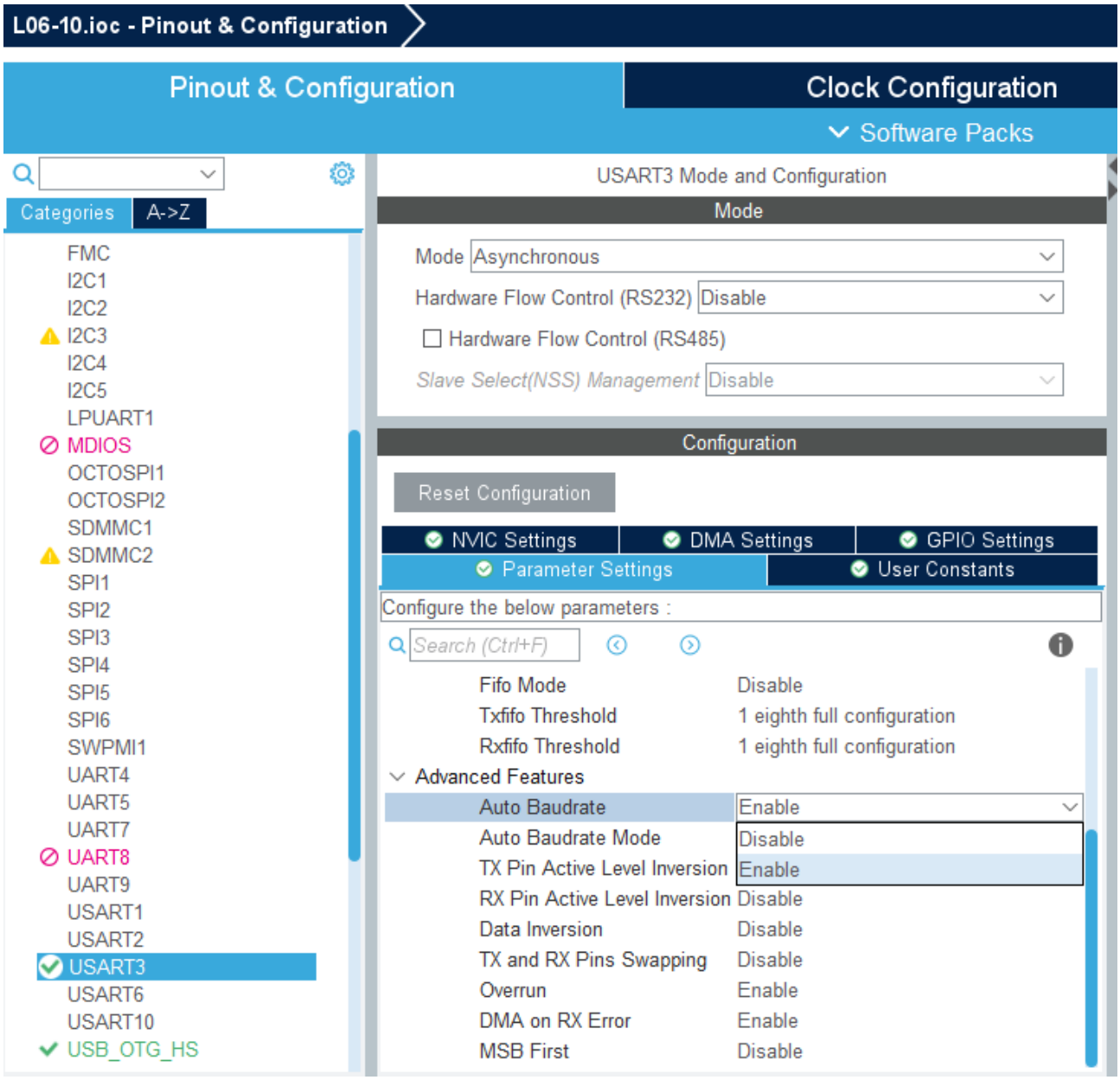
Uwaga na Linux (przynajmniej Debian). Przy prędkości 460800 lub wyższej, przerwy pomiędzy bajtami są dłuższe niż 1,5 znaku. Nie jest to wina sprzętu bo na tym samym komputerze z uruchomionym Win7 znaki były wysyłane bez zbędnych przerw. Problemem nie tyło też ustawienie terminala w Linux, który standardowo pomiędzy znakami wstawiał opóźnienie, ta opcja była wyłączona. To pierwszy raz kiedy Linux mnie rozczarował :-(

???? zrobić test na Rpi (analizator w ruch)

6.10 Autobaud

Ta funkcja to marzenie AVR'owców. Co prawda da się ją uzyskać korzystając z przerwań zewnętrznych i timera ale wymaga to dodatkowego nakładu pracy i wyliczeń zależnych od taktowania CPU. W STM32 wszystko dzieje się automatycznie.

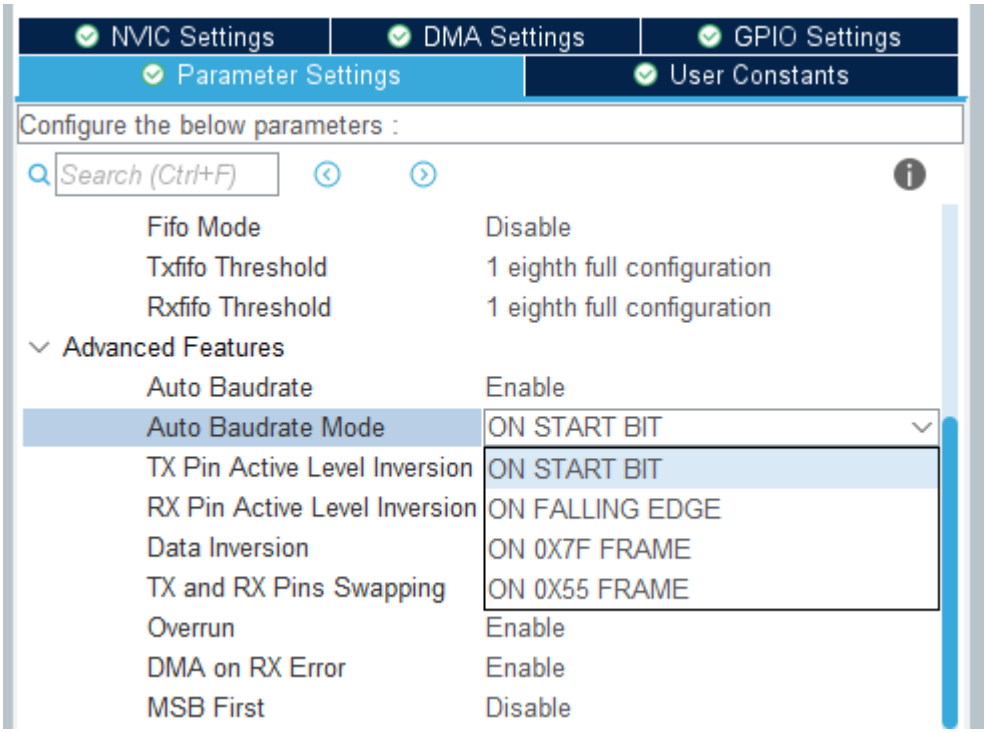
Pominę konfigurację UART, która jest typowa. Włączenie funkcjonalności autodetekcji:



Rysunek r06-10_1

Uwaga! Nie wszystkie UART,y mają tą funkcję mimo, że w CubeMX on występuje. Reguła jest taka, że UART'y o niższych numerach są bogato wyposażone te o wyższych niekoniecznie. Szczegółów trzeba szukać w dokumentacji mikrokontrolera.

Tryb pracy zostawiamy domyślny:



Rysunek r06-10_2

Film z wyjaśnieniami: https://youtu.be/PEGqhvNGwOs

Projekt: L06-10

6.11 Drukarka termiczna

Do zabawy wybrałem EM5820, która posiada interfejsy: RS232C, UART i USB. Napięcie zasilania 5-9V 1,5A. Drukarkę kupiłem na Allego za 75zł (85 z kosztami wysyłki, krótko po zakupie cena wzrosła do 90zł. Na drukarkę czekałem niecałe dwa tygodnie (12 dni).

Dawniej takie ,drukarki miały dużo zastosowań jak wszelkiej maści rejestratory, osobiście widziałem w roli rejestratora rozmów (bilingów) w centrali abonenckiej (PABX) Panasonica, do wydruku rachunków przez inkasentów. Teraz w roli rejestratora pracuje pamięć FLASH w urządzeniu albo na karcie SD czy pendrive.

Teraz drukarki termiczne można spotkać w kasach fiskalnych, parkomatach czy popularnych ostatnio Butelkomatach.

Podłączenie:

Zasilanie. Nie można zasilić przez złącze USB, zasilanie należy doprowadzić do do dedykowanego złącza lub RS232C albo TTL. Drukarka w czasie wydruku pobiera ponad 1,3A producent zaleca

zasilacz 1,5A

TX drukarki jest linią wejściową! Łączymy ją z TX UART'a.

????? schemat

Kody sterujące A2-user-manual drukarki.

?????

Drukowanie jest procesem długotrwałym dlatego zaleca się dane do wydruku umieścić w pamięci RAM (mamy jej dużo to nie AVR) a z niej wysłać używając DMA.

Przykładowe nieblokujące funkcje drukowania pokazano na listingu:

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */


uint16_t timLED, timPrinterStart = 1000;
void HAL_IncTick(void){
	uwTick += (uint32_t)uwTickFreq;

	if ( timLED ) timLED--;
	if ( timPrinterStart && timPrinterStart != UINT16_MAX ) timPrinterStart--;

}


typedef enum{PRT_INITED, PRT_INIT, PRT_IDLE, PRT_PRINTING } printerStatus_dt;
printerStatus_dt volatile printerStatus;
HAL_StatusTypeDef printerInit(){
	const uint8_t prtInitTxt[]= {27,64,          // Init
	};

	return HAL_UART_Transmit_DMA(&huart1, prtInitTxt, sizeof(prtInitTxt) );
}


#define	PRT_BUF_SIZE 10000
uint16_t printerBufLen;
uint8_t printerBuffer[PRT_BUF_SIZE];
HAL_StatusTypeDef printerBufAdd(uint8_t* data, uint16_t len){

	if( printerBufLen + len < PRT_BUF_SIZE ){
		memcpy(printerBuffer + printerBufLen, data, len);
		printerBufLen += len;
		return HAL_OK;
	}

	return HAL_ERROR;
}
```

```
HAL_StatusTypeDef printerBufPrint(){
	if( printerStatus == PRT_INIT || printerStatus == PRT_IDLE ){
		printerStatus = PRT_PRINTING;
		return HAL_UART_Transmit_DMA(&huart1, printerBuffer, printerBufLen );
	}

	return HAL_ERROR;
}


void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart){
	if ( huart == &huart1 ){
		if ( printerStatus == PRT_INITED ) printerStatus++;
		if ( printerStatus == PRT_PRINTING ){
			printerBufLen = 0;
			printerStatus = PRT_IDLE;
		}
	}

}

/* USER CODE END 0 */
```

Inicjalizacja drukarki:

```
	/* USER CODE BEGIN 2 */


	printerInit();


	/* USER CODE END 2 */
```

Wydruk przykładowego tekstu:

```
		if ( !timPrinterStart && printerStatus == PRT_INIT ){
			timPrinterStart = UINT16_MAX;

			const char prtString1[] = "Hello, EM5820 Thermal Printer!\n";
			printerBufAdd( (uint8_t*)prtString1, strlen(prtString1) );

			const uint8_t prtBoldOn[] = {27,69,1};
			printerBufAdd( (uint8_t*)prtBoldOn, sizeof(prtBoldOn) );

			const char prtString21[] = "Bold.\n";
			printerBufAdd( (uint8_t*)prtString21, strlen(prtString21) );

			const uint8_t prtBoldOff[] = {27,69,0};
			printerBufAdd( (uint8_t*)prtBoldOff, sizeof(prtBoldOff) );

			const uint8_t prtDoubleHeight[] = {27,33, 16};
```

```
                printerBufAdd( (uint8_t*)prtDoubleHeight, sizeof(prtDoubleHeight) );

                const char prtString2[] = "Nucleo STM32H723ZG.\n";
                printerBufAdd( (uint8_t*)prtString2, strlen(prtString2) );

                const uint8_t prtDoubleWidth[] = {27,33, 32};
                printerBufAdd( (uint8_t*)prtDoubleWidth, sizeof(prtDoubleWidth) );

                const char prtString3[] = "Double Width\n";
                printerBufAdd( (uint8_t*)prtString3, strlen(prtString3) );

                const uint8_t prtDoubleWH[] = {27,33,32+16};
                printerBufAdd( (uint8_t*)prtDoubleWH, sizeof(prtDoubleWH) );

                const char prtString4[] = "Double W+H\n";
                printerBufAdd( (uint8_t*)prtString4, strlen(prtString4) );

                printerBufAdd( (uint8_t*)prtBoldOn, sizeof(prtBoldOn) );
                const char prtString5[] = "Double W+H+B\n";
                printerBufAdd( (uint8_t*)prtString5, strlen(prtString5) );

                const uint8_t prtDoubleOff[] = {27,33, 0};
                printerBufAdd( (uint8_t*)prtDoubleOff, sizeof(prtDoubleOff) );
                printerBufAdd( (uint8_t*)prtBoldOff, sizeof(prtBoldOff) );


                const char prtLine[] = "--------------------------------\n";
                printerBufAdd( (uint8_t*)prtLine, strlen(prtLine) );

                const char prtFP[] = {27, 100, 5};           // Line Feed 27, 100 n
                printerBufAdd( (uint8_t*)prtFP, strlen(prtFP) );

                __NOP();
                printerBufPrint();

        }
```

Kod nie jest doskonały, bo gdy chcemy dodać coś do wydruku w czasie jego trwania otrzymamy błąd. Problem rozwiązałby mechanizm kolejki który opisałem w książce dla zaawansowanych.

Cały kod (projekt): L06-11

Uwaga! Po rozpakowaniu nazwa projektu „**H06-11**” a nie „L06-11”

Film: https://youtu.be/R1qO45tyVNI

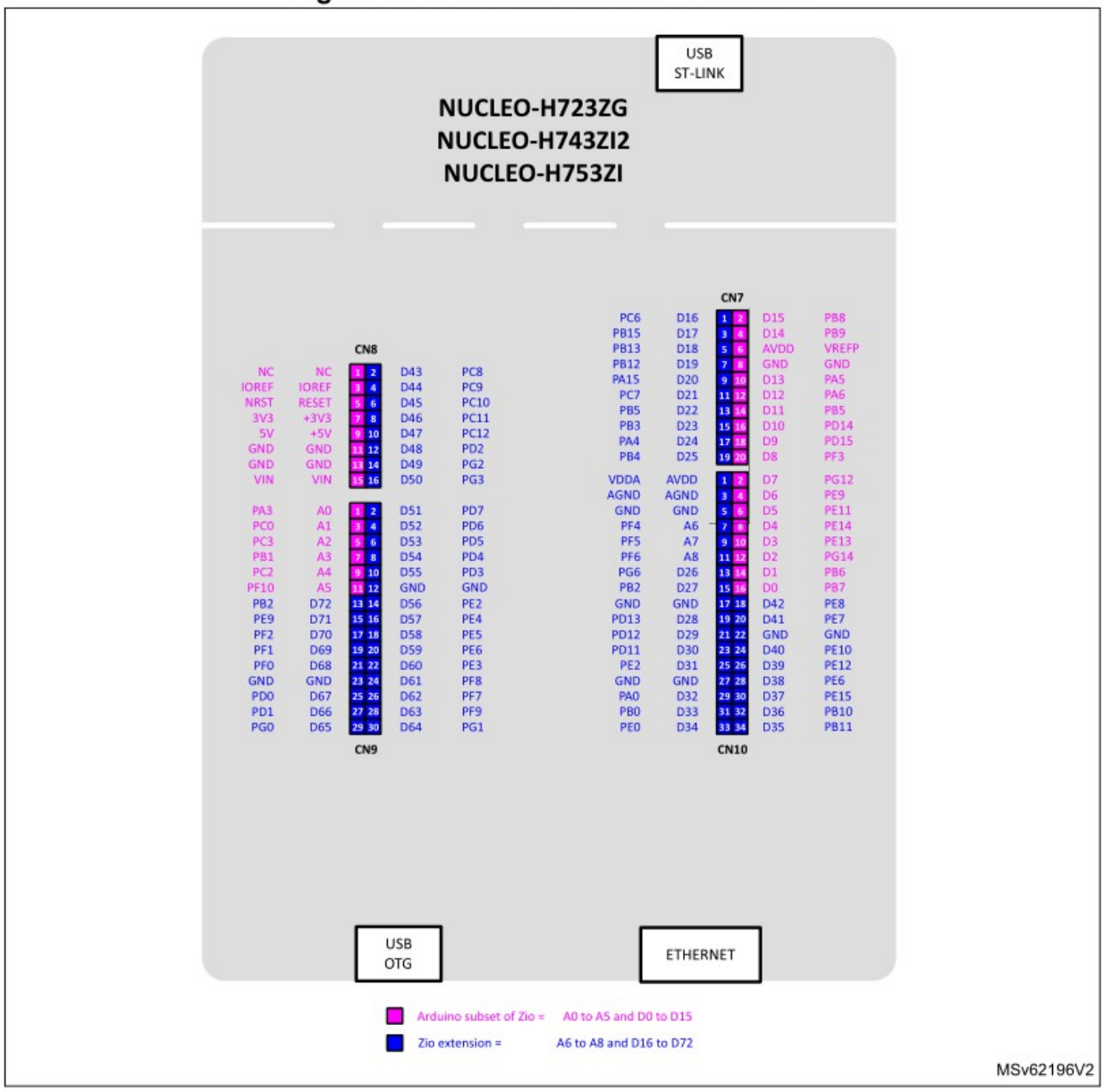
Rozdział 7. I2C

Czym jest I2C i jak działa? W Internecie jest dużo materiałów na ten temat więc nie widzę sensu aby je tu powielać. Zainteresowani mogą sobie poczytać:

Wiki, Msalamon, Botland

Po teorii przejdziemy do praktyki. Testy przeprowadzimy z układem PCF8574, który znajduje się na płytce konwertera I2C do alfanumerycznego LCD. PCF8574 nie jest zbyt dobry ale z nim są popularne i tanie. Z dokumentacji płytki NUCLEO:

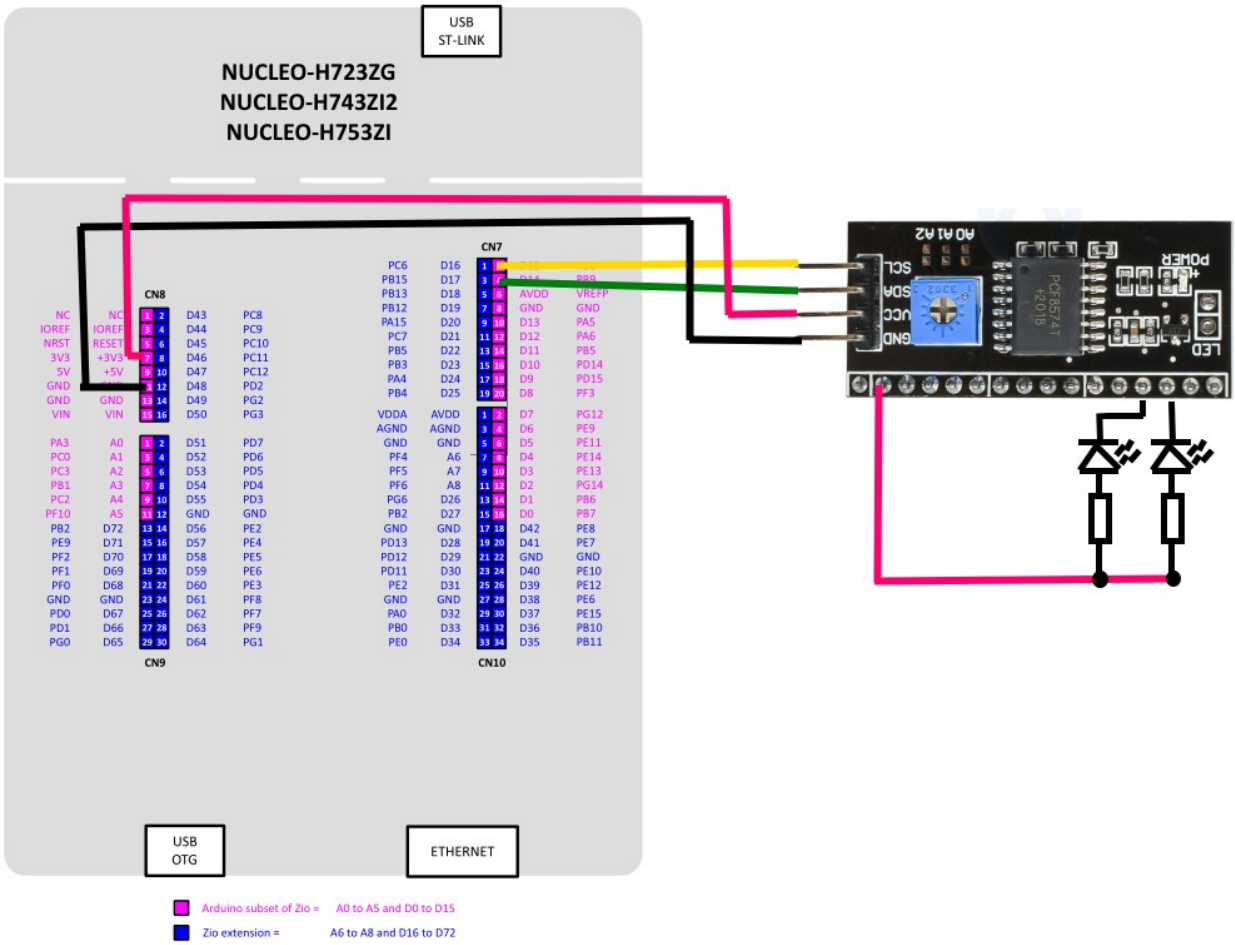
Figure 16. NUCLEOH7 Nucleo-144 board



Rysunek r07-0

Dowiemy się, które GPIO przypisano do linii SCL (D15) i SDA (D14). Przeglądając schematy dojdziemy do wniosku, że li nie I2C nie są podciągnięte do zasilania. Wbudowane w STM32 podciąganie jest za słabe aby zapewnić poprawną komunikację choćby przy zegarze 100kHz. Szczęśliwie większość modułów dla Arduino ma wbudowane podciąganie, zwykle jest to 10k ale w przypadku modułu który posiadam jest to 4,7k. Gdyby podciągania nie było to musimy dodać rezystory albo zmniejszyć szybkość taktowania I2C do 10kHz lub mniej.

Schemat połączeń:

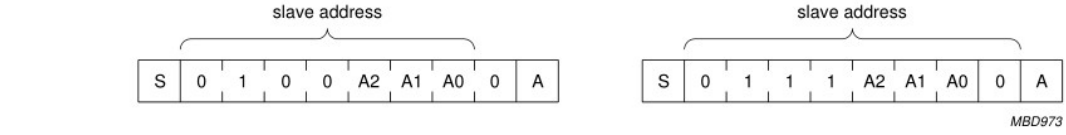


Rysunek r07-1

Układ PCF8574 występuje w dwóch wariantach różniący się adresem:

7.1 Addressing

For addressing see Figs 9, 10 and 11.



(a) PCF8574.
(b) PCF8574A.

Fig.9 PCF8574 and PCF8574A slave addresses.

Rysunek r07-2

Układy różnią się stałą częścią adresu 0100 bez litery i 0111 z literą A. W połączeniu z możliwością sprzętowego ustawienia trzech najmłodszych bitów adresu pozwala to na podłączenie 16 układów do jednej magistrali. W płytkach dla Arduino adres zwykle jest ustawiony na 111 a to oznacza, że PFC8574 ma adres 0x27 a PCF8574A 0x3F. Te adresy są prawdziwe dla funkcji przyjętych w Arduino (7-bit), HAL dla STM32 przyjmuje adresowanie przedstawione na rysunku r07-2. PCF8574 ma więc adres 0x4E a PCF8574A 0x7E. Układ akceptuje zasilanie napięciem od 2,5 do 6V.

7.1 Zapis

Jest bardzo prostu, realizuje go funkcja:

```
HAL_I2C_Master_Transmit_IT
```

lub gdy chcemy użyć DMA:

```
HAL_I2C_Master_Transmit_DMA
```

7.2 Odczyt

Tak banalny jak i zapis:

```
HAL_I2C_Master_Receive_DMA
```

z tym, że dana jest odczytana po wywołaniu:

```
HAL_I2C_MasterRxCpltCallback
```

Kod: L07-2

Film: https://youtu.be/JAMU8i1pqEE

7.3 Skaner I2C, wyszukiwanie układów na magistrali

Funkcje potrzebne do skanowania:

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

void printDBG(char *txt){
	HAL_UART_Transmit(&huart3, (uint8_t*)txt, strlen(txt), 10);
}


void I2C_Scan(I2C_HandleTypeDef *hi2c) {
	printDBG("Find I2C...\r\n");
	// Zakres adresów na magistrali I2C (7-bitowe adresy)
	for (uint8_t address = 1; address < 128; address++) {
		// Wysyłanie zapytania do adresu (operacja "ping")
		if (HAL_I2C_IsDeviceReady(hi2c, (address << 1), 1, 10) == HAL_OK){
			char txt[50];
			sprintf(txt,"0x%02X (0x%02X) ", address, address<<1);
			printDBG(txt);
		}
	}
	printDBG("End.\r\n");
}

/* USER CODE END 0 */
```

Wywołanie:

```
  /* USER CODE BEGIN 2 */

	I2C_Scan(&hi2c1);

  /* USER CODE END 2 */
```

Film: https://youtu.be/xECzEdnsHpE
Kod: L07-3

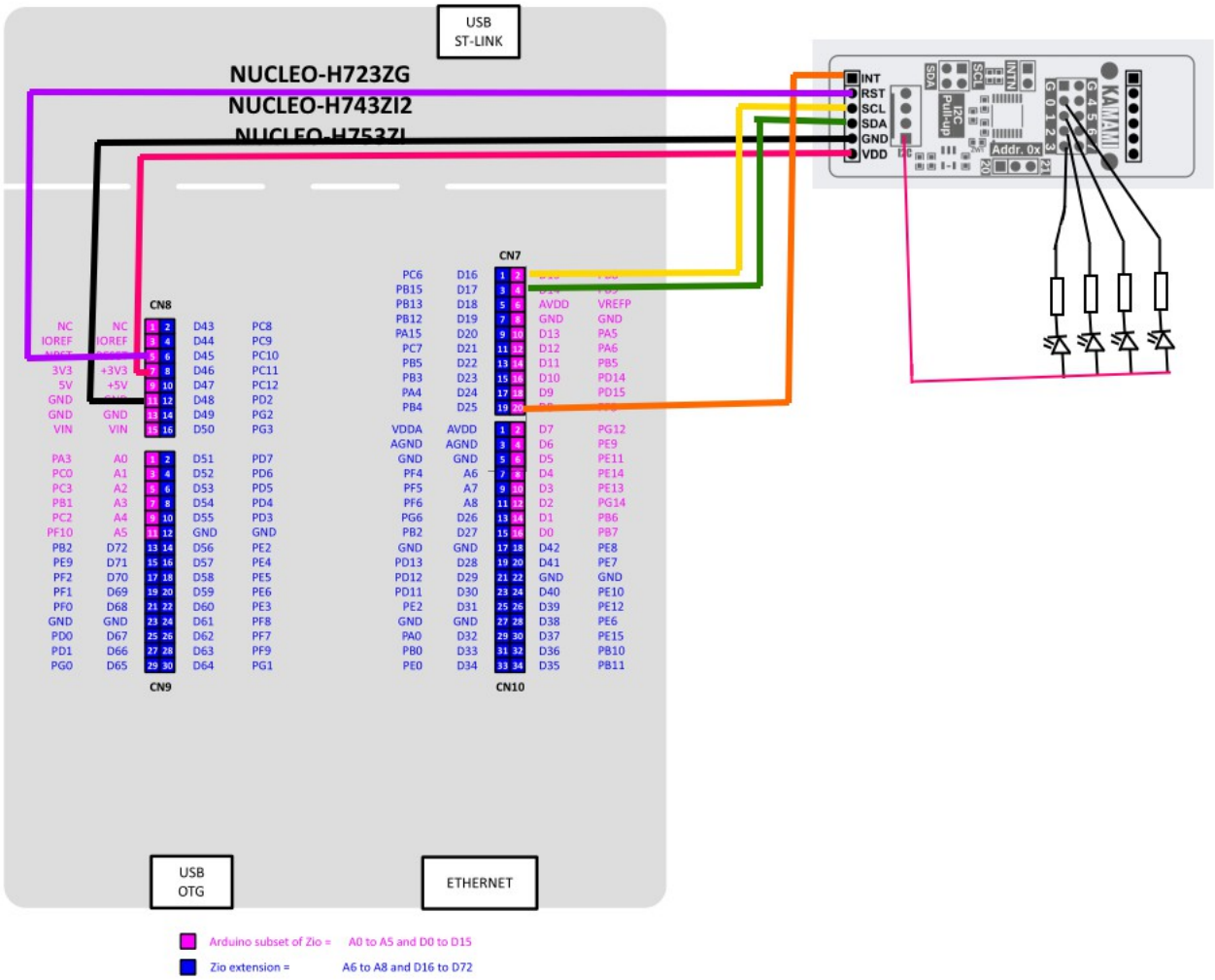
7.4 Ekspander PCAL6408

Skorzystam z modułu: KamodPCAL6408, Który otrzymałem od firmy https://kamami.pl.
Dokumentacja modułu, z którą warto się zapoznać przed lekcją.

7.4.1 Proste GPIO

7.4.1.1 Output

Na początek zamigamy sobie LED'ami. Schemat połączeń:



Rysunek r07-4

Kod programu:

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

#define      PCAL6408_ADR        0x40
#define      PCAL6408_OUTPORT    0x01
#define      PCAL6408_CFG 0x03

/* USER CODE END Includes */
```

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint16_t volatile timPCAL6408 = 10;
void HAL_IncTick(void){
uwTick += (uint32_t)uwTickFreq;


      if ( timPCAL6408 ) timPCAL6408--;
}


/* USER CODE END 0 */
```

```
  /* USER CODE BEGIN 2 */

  uint8_t cfg = 0b11110000;
  HAL_I2C_Mem_Write_IT(&hi2c1, PCAL6408_ADR, PCAL6408_CFG, 1, &cfg, 1 );


  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1){
         HAL_IWDG_Refresh(&hiwdg1);
         __WFI();

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

         if ( !timPCAL6408 ){
                timPCAL6408 = 500;

                uint8_t static pData = 0x55;
                HAL_I2C_Mem_Write_IT(&hi2c1, PCAL6408_ADR, PCAL6408_OUTPORT, 1, &pData, 1 );

                pData ^= 0xFF;
         }


  }
  /* USER CODE END 3 */
```

Film: https://youtu.be/q1-v4b7ccGA

Projekt: L07-4

7.4.1.2 Input

Odczyt pinów wejściowych, kod:

```
/* USER CODE BEGIN Includes */

#define      PCAL6408_ADR        0x40
#define      PCAL6408_INPUTPORT  0x00
```

```
#define      PCAL6408_OUTPORT    0x01
#define      PCAL6408_CFG        0x03
#define      PCAL6408_PULLEN     0x43
#define      PCAL6408_PULLSEL    0x44

#include <stdio.h>

/* USER CODE END Includes */
```

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

uint16_t volatile timPCAL6408 = 10;
void HAL_IncTick(void){
      uwTick += (uint32_t)uwTickFreq;


      if ( timPCAL6408 ) timPCAL6408--;
}


uint8_t PCAL6408_input;

void PCAL_init(){
      uint8_t cfg;
      cfg = 0xFF;         //włącz pullup (sprawdzic miltimetrem)
      HAL_I2C_Mem_Write(&hi2c1, PCAL6408_ADR, PCAL6408_PULLEN, 1, &cfg, 1, 10 );    //
Operacje blokujące w inicie są dopuszczalne
      cfg = 0b11110000;
      HAL_I2C_Mem_Write(&hi2c1, PCAL6408_ADR, PCAL6408_CFG, 1, &cfg, 1, 10 );
}


/* USER CODE END 0 */
```

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)  {
      HAL_IWDG_Refresh(&hiwdg1);
      __WFI();

      /* USER CODE END WHILE */

      /* USER CODE BEGIN 3 */

      if ( !timPCAL6408 ){
            timPCAL6408 = 20;               // 50Hz

            HAL_I2C_Mem_Read(&hi2c1, PCAL6408_ADR, PCAL6408_INPUTPORT, 1, &PCAL6408_input, 1, 10 );

            uint8_t static in;
            if ( in != PCAL6408_input ){
                  in = PCAL6408_input;

                  char static txt[30];
                  sprintf(txt,"in=%x ", PCAL6408_input);
                  HAL_UART_Transmit_IT(&huart3, (uint8_t*)txt, strlen(txt) );
            }
```

```
        }


}
/* USER CODE END 3 */
```

Film: https://youtu.be/Rj5osHyNBTE

Projekt: L07-4.2

Niestety w kodzie w pętli głównej korzystamy z funkcji blokujących do obsługi I2C. Naprawia to kolejny kod (pokazano tylko zmiany):

```
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

uint16_t volatile timPCAL6408 = 10;
void HAL_IncTick(void){
        uwTick += (uint32_t)uwTickFreq;
        if ( timPCAL6408 ) timPCAL6408--;
}


uint8_t PCAL6408_input;
void PCAL_init(){
        uint8_t cfg;
        cfg = 0xFF; //włącz pullup (sprawdzic miltimetrem)
        HAL_I2C_Mem_Write(&hi2c1, PCAL6408_ADR, PCAL6408_PULLEN, 1, &cfg, 1, 10 ); // Operacje blokujące w
inicie są dopuszczalne
        cfg = 0b11110000;
        HAL_I2C_Mem_Write(&hi2c1, PCAL6408_ADR, PCAL6408_CFG, 1, &cfg, 1, 10 );
}


HAL_StatusTypeDef printUART(char *txt){
        return HAL_UART_Transmit_DMA(&huart3, (uint8_t*)txt, strlen(txt) );
}


void HAL_I2C_MemRxCpltCallback(I2C_HandleTypeDef *hi2c){
        /* Prevent unused argument(s) compilation warning */
        UNUSED(hi2c);

        uint8_t static in;
        if ( in != PCAL6408_input ){
                in = PCAL6408_input;
                char static txt[50];
                sprintf(txt,"in=%x ", PCAL6408_input);
                printUART( txt );
                uint8_t PCAL6408_output = PCAL6408_input >>= 4;
                HAL_I2C_Mem_Write_IT(&hi2c1, PCAL6408_ADR, PCAL6408_OUTPORT, 1, &PCAL6408_output, 1 );
        }
}


/* USER CODE END 0 */
```

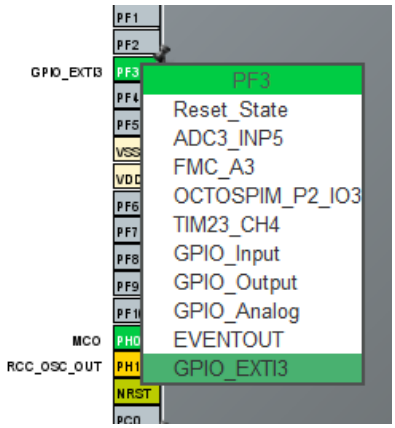
```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1){
        HAL_IWDG_Refresh(&hiwdg1);
        __WFI();

        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
        if ( !timPCAL6408 ){
```

```
                timPCAL6408 = 20; // 50Hz
                HAL_I2C_Mem_Read_IT(&hi2c1, PCAL6408_ADR, PCAL6408_INPUTPORT, 1, &PCAL6408_input, 1 );
        }
}
/* USER CODE END 3 */
```

Film: https://youtu.be/FsWBHJZJL_s
Projekt: L07-4.3

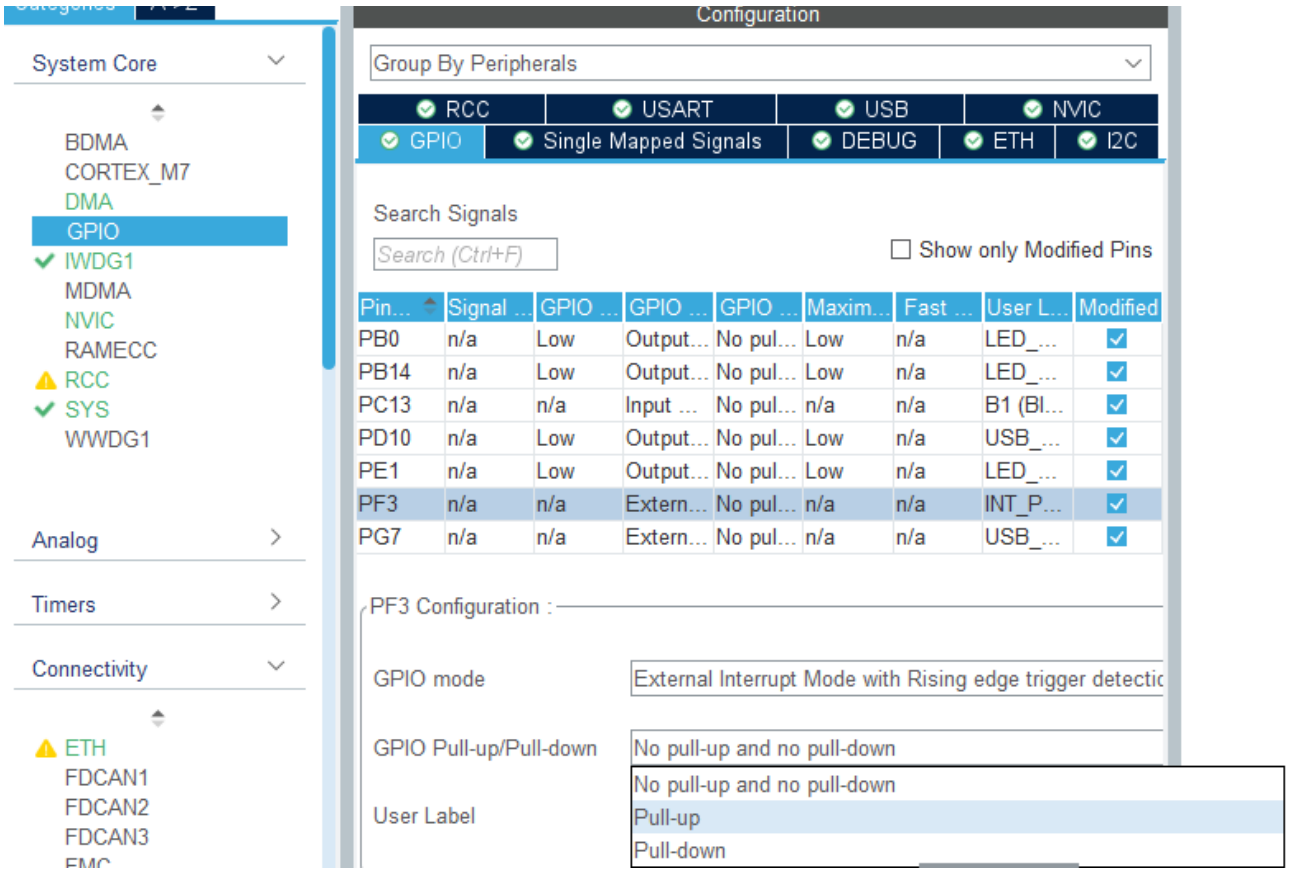
7.4.2 Przerwania

Poprzedni kod na przerwaniach jest wydajniejszy od wcześniejszych ale nie jest idealny. Co 20ms odczytujemy rejestry ekspandera. Czy można coś z tym zrobić? Można! Skonfigurujemy ekspander tak aby generował przerwania gdy zmieni się stan portu wejściowego. STM32 zareaguje na takie przerwanie odczytem portu wejściowego ekspandera, reszta kodu pozostaje taka sama. W CubeMX trzeba pamiętać aby linie GPIO połączone z wyjściem przerwań ekspandera ustawić w tryb EXTI:



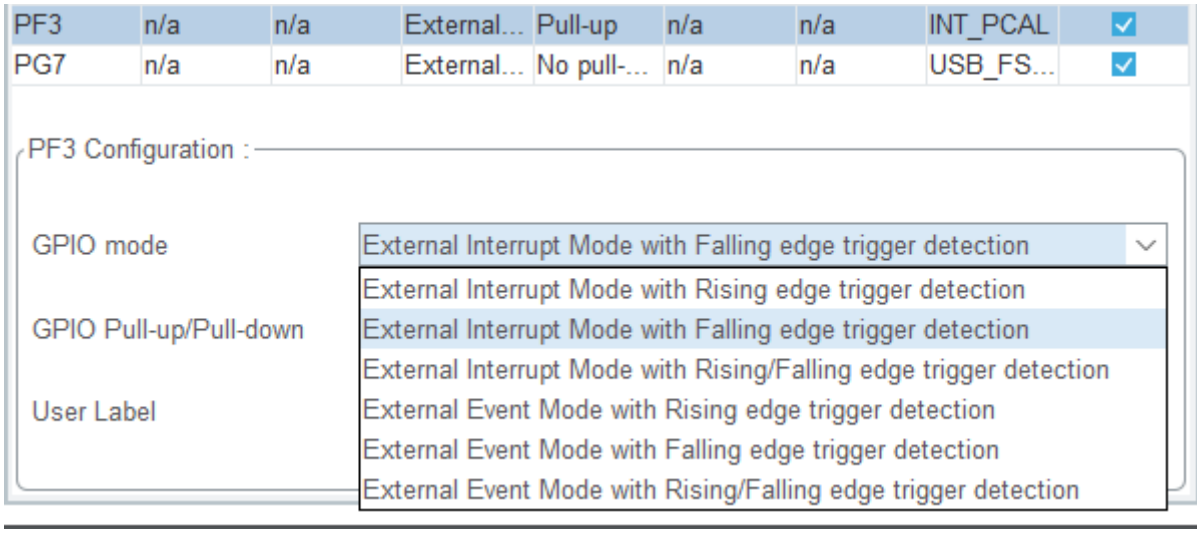
Rysunek r07-4_2_1

oraz podciąganie wejścia:



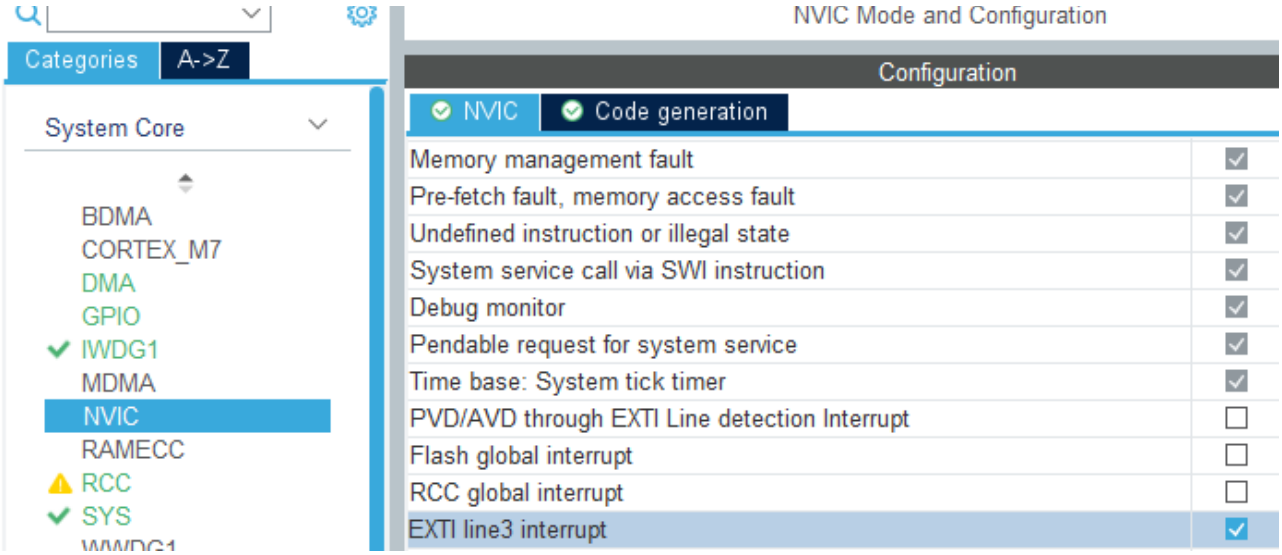
Rysunek r07-4_2_2

Trzeba pamiętać aby ustawić przerwania od zbocza opadającego:



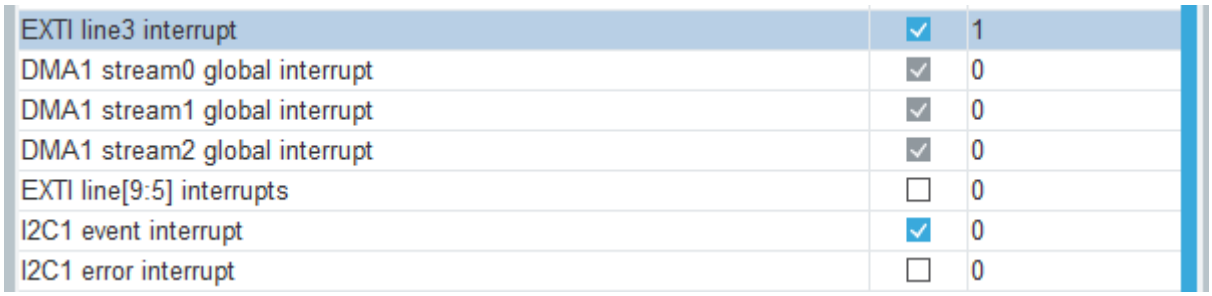
Rysunek r07-4_2_3

a także włączyć same przerwania od wybranego EXTI:



Rysunek r07-4_2_3

Istotne jest aby przerwanie EXTI miało niższy priorytet (wyższą wartość) niż przerwanie I2C. Domyślnie I2C ma najwyższy zero więc EXTI ustawimy na 1:



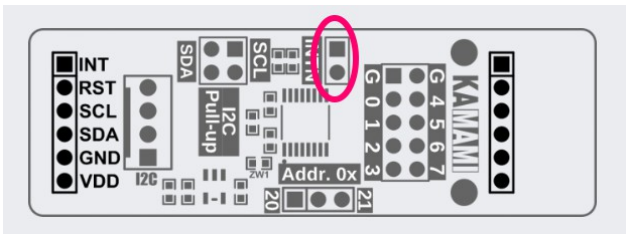
Rysunek r07-4_2_4

To jednak nie wystarczy aby przerwania od ekspandera działały, bo trzeba je w nim włączyć:

```
    cfg = 0b00001111;   // włączenie IRQ (0-włącza, 1-wyłącza)
    HAL_I2C_Mem_Write(&hi2c1, PCAL6408_ADR, PCAL6408_INTMSK, 1, &cfg, 1, 10 );
```

Uwaga!

Standardowo zworka INT na module KamodPCAL6408 nie jest założona!



Rysunek r07-4_2_5

Obsługa przerwania EXTI:

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
    if ( GPIO_Pin == INT_PCAL_Pin ) {
        __NOP();
        // Odczyt INPUT skasuje IRQ
                HAL_I2C_Mem_Read_IT(&hi2c1, PCAL6408_ADR, PCAL6408_INPUTPORT, 1, &PCAL6408_input, 1 );
    }
}
```

Film z wyjaśnieniami: https://youtu.be/yOjLKvapP0E

Cały kod programu: L07-4.4

Ciekawszy wydaje mi się układ MCP23017. Jego wersję SPI (MCP23S17) opisałem w książce dla zaawansowanych. Kilka filmów:

film 70a: https://youtu.be/j85Ozl2Bcd8

film 70b: https://youtu.be/h3rm_UpgU2E

Wersja I2C:

Film „Obsługa przerwań od MCP23017”: https://youtu.be/c6yiF7CjZzc

Film „Lepsza alternatywa dla PCF8574, czyli o MCP23017”: https://youtu.be/LkSjIGBrsQQ

Film „Jeszcze o MCP23017 część 2 - sprostowania, dodatkowe wyjaśnienia”:

https://youtu.be/yhRN_v8kklA

7.4.3 Klawiatura matrycowa

Idea: https://youtu.be/jNiULFCv-T8

?????

Ciach... Więcej w pełnej wersji książki.

7.5 EESRAM 47Lxx

Przeciwnicy STM32 zarzucają im brak pamięci EEPROM. To bzdura wynikająca z niewiedzy lub manipulacji faktami, bo są STM32 z EEPROM przykładowo STM32L152. Pamięć EEPROM zwykle łatwo zaimplementować w FLASH co opisałem szczegółowo z książce Kurs programowania STM32.pdf. Taka emulowana pamięć ma wiele zalet w stosunku do wbudowanej. W ostateczności można użyć pamięci zewnętrznej I2C, SPI lub gdy mamy mało GPIO 1-Wire. Jednak pamięci EEPROM mają istotna wadę, długi czas zapisu a ściślej kasowania. Wady tej nie maja FRAM ale są drogie. Można użyć pamięć SRAM podtrzymywanej baterią ale jest ciekawsze rozwiązanie EESRAM. Jest to połączenie pamięci SRAM z EEPROM. Po załączeniu zasilania zawartość EEPROM kopiowana jest do SRAM. W czasie normalnej pracy używamy SRAM a więc pamięci szybkiej bez ograniczenia liczby zapisów. Poi wyłączeniu zasilania zawartość SRAM jest

kopiowana do EEPROM. Energia do tej operacji pobierana jest z zewnętrznego kondensatora. Proces kopiowania pomiędzy pamięciami odbywa się automatycznie ale można go robić „ręcznie” gdy przykładowo urządzenie często jest wyłączane co degraduje EEPROM zapisami do niej. Cena pamięci jest atrakcyjna trochę wyższa niż EEPROM ale dużo niższa od FRAM. Aby nie było zbyt dobrze EESRAM też mają wady, którą jest stosunkowo mała pojemność. Na chwilę pisania tej książki największa dostępna pojemność w sprzedaży detalicznej to 2kB.

7.6 OLED 64x32

7.7 Pseudo I2C – sterownik wyświetlacza i klawiatury TM1637

7.8 Kalkulator z drukarką

Klawiatura matrycowa z PCAL6408

Wyświetlacz z TM1637

Drukarka termiczna

Rozdział 8. SPI

?????

Sterowanie linią SS (przerwanie od końca transferu)

Wyświetlacz TFT 320x240

Własne funkcje graficzne.

Rysowanie punktu

Rysowanie linii, prostokątów

Teksty

Rozdział 9. USB

Rozdział 10. 1-Wire

Przez GPIO (Blokujące)

Przez UART (na IRQ)

DS18B20

Rozdział 11. RTC

Rozdział 12. ADC

Rozdział 13. DAC

Rozdział 14. Odczyt pilota IR

Rozdział 15. WS2812

Rozdział n....

Sterowanie wyświetlaczem LCD 7-segmentów

Na „piechotę” używając GPIO

Używając sterownika wbudowanego w STM32

????? dobrać NUCLEO

Alfanumeryczny LCD ze sterownikiem HD6xxxxx

Rozdział n+1. Budujemy mówiący zegarek z budzikiem

Rozdział n+2. Budujemy stację pogodową

Rozdział n+3. Waga kolejowa do modeli

Tesometr + wyświetlacz multipleksowany albo OLED

Rozdział n+4. Generator PWM

Patroni (wspierający) powstanie książki

Pomioty gospodarcze:

https://kamami.pl

Osoby prywatne:

Daniel Piłat , harold haroldps, Krzysztof Mytnik, Krzysztof Szwaba, Pawel Wozniak, Artur Skoneczny, Tomasz Krząkała, Robert Loboda, Cezary Zawadzki,
Grzegorz Sosnowski, Jakub Pilch, Kamil Szafraniec

Okładka 2

O autorze:

Absolwent ZSZ w Zespole Szkół Elektrycznych we Włocławku. Od 1991 do końca 2001 roku pracował w TPSA na stanowisku Instruktora. Od początku 2002 roku w Zjednoczonych Przedsiębiorstwach Rozrywkowych zajmował się zawodowo projektowaniem urządzeń i pisaniem programów na nie. Urządzenia wykorzystują głównie mikrokontrolery czasem mikroprocesory i/lub układy programowalne GAL, CPLD, FPGA. Od 1997 do 2022 roku współpracował z wydawnictwem AVT publikując około 250 artykułów. Publikował także w „Magazynie Amiga", „Praktycznym Elektroniku", „Hobby Elektroniku". W 1991 roku w „Praktycznym Elektroniku" pojawia się pierwszy artykuł, kiedy to miał 17 lat w chwili publikacji (projekt z 1989 roku). Od 1991 pisał programy i budował urządzenia dla Commodore 64, od 1993 dla Amigi, w latach 1993-1994 (podczas odbywania ZSW) opracował kilka konstrukcji dla MON (na TTL, C-MOS i 6502). Pierwsze recenzje programów i urządzeń dla C-64 i Amigi pojawiły się w 1993 roku w „Commodore & Amiga". Hobbystycznie zajmuje się modelarstwem (głównie kolejowym) z naciskiem na elektronikę. Efektem hobby jest tani dekoder trakcji i dźwięku sterowany przez DCC http://kolejki.prv.pl/dekoderySTM32 oraz urządzenia SRK. Od 2019 roku prowadzi kanał na YouTube https://youtube.com/@eR-MIK o tematyce elektronicznej poruszający głównie zagadnienie związane z mikrokontrolerami, mikroprocesorami i układami programowalnymi.

Skąd pomysł na książkę? ?????

Kontakt z autorem: sas.ze@vp.pl

Kanał na YouTube: https://youtube.com/@eR-MIK

Strony internetowe autora:
http://er-mik.prv.pl/ - strona PHP/HTML bez użycia kreatorów (tylko edytor tekstu ASCII pierwowzór strony stworzony na Amidze)
http://es2.noip.pl - strona na serwerze autora
http://kolejki.prv.pl/ - tematyka modelarstwa kolejowego

Portfolio autora:
http://er-mik.prv.pl/projekty_avt.php - projekty dla AVT, MA, PE, HE, BIW, FET od 1991 roku
http://er-mik.prv.pl/projekty_edw.php - projekty dla AVT po 2017 roku
http://er-mik.prv.pl/projekty_ze.php - projekty dla „Zrozumieć elektronikę"