

POLITECHNIKA POZNAŃSKA

WYDZIAŁ ELEKTRYCZNY

INSTYTUT AUTOMATYKI, ROBOTYKI I INŻYNIERII

INFORMATYCZNEJ

ZAKŁAD STEROWANIA I ELEKTRONIKI PRZEMYSŁOWEJ



PROJEKT

MANIPULATOR STEROWANY NATĘŻENIEM OŚWIETLLENIA

MACIEJ SADOWSKI, ADAM KOŁODZIEJCZYK

SYSTEMY MIKROPROCESOROWE

PROJEKT

PROWADZĄCY:

MGR INŻ. ADRIAN WÓJCIK

POZNAŃ, 10.02.2020R.

DANE SZCZEGÓŁOWE

Rok studiów: INŻ. 3

Rok akademicki: 2019/2020

Termin zajęć: środa g. 8:00

Data wykonania: 12.02.2019

Temat zajęć: Manipulator sterowany natężeniem oświetlenia

Prowadzący: mgr inż. Adrian Wójcik

Skład grupy (nazwisko, imię, nr indeksu):

1. Sadowski Maciej 135886
2. Kołodziejczyk Adam 135840

OCENA (*WYPEŁNIA PROWADZĄCY*)

Spełnienie wymogów redakcyjnych:

.....

Wykonanie, udokumentowanie oraz opis wykonanych zadań:

.....

Zastosowanie prawidłowego warsztatu programistycznego:

.....

Spis treści

1.	Wstęp	4
2.	Konstrukcja mechaniczna i elektryczna układu	4
2.1	Konstrukcja mechaniczna	4
2.2	Układ elektroniczny	5
3.	Sterowanie urządzeniami wykonawczymi	5
3.1	Specyfikacja	5
3.2	Implementacja sterowania diodami LED	5
3.3	Implementacja sterowania Serwomechanizmem	5
3.4	Implementacja sterowania urządzeń wyjściowych	6
3.5	Wyniki Testów	7
3.6	Podsumowanie	7
4.	Pomiar natężenia światła	7
4.1	Specyfikacja	7
4.2	Implementacja	7
4.3	Wyniki Testów	8
4.4	Podsumowanie	8
5.	Obsługa komunikacji szeregowej wraz z kontrolą błędów	8
5.1	Specyfikacja	8
5.2	Implementacja	8
5.3	Wyniki Testów	11
5.4	Podsumowanie	11
6.	Algorytm Sterowania	11
6.1	Specyfikacja	11
6.2	Implementacja	11
6.3	Wyniki Testów	13
6.4	Podsumowanie	15
7.	Dedykowany program okienkowy	15
7.1	Specyfikacja	15
7.2	IMPLEMENTACJA	15
7.2.1	Interfejs graficzny	15
7.3	Wyniki testów	18
7.4	Podsumowanie	19
8.	Literatura	19
9.	Załączniki	19

1. WSTĘP

Celem projektu jest zbudowanie układu regulacji manipulatora w postaci obrotowego ramienia, sterowanego zmianami natężenia światła. Manipulator znajduje się w zamkniętej przestrzeni, odizolowanej od kontaktu ze światłem zewnętrznym. W przestrzeni tej umieszczone są punktowe źródła światła.

2. KONSTRUKCJA MECHANICZNA I ELEKTRYCZNA UKŁADU

2.1 KONSTRUKCJA MECHANICZNA

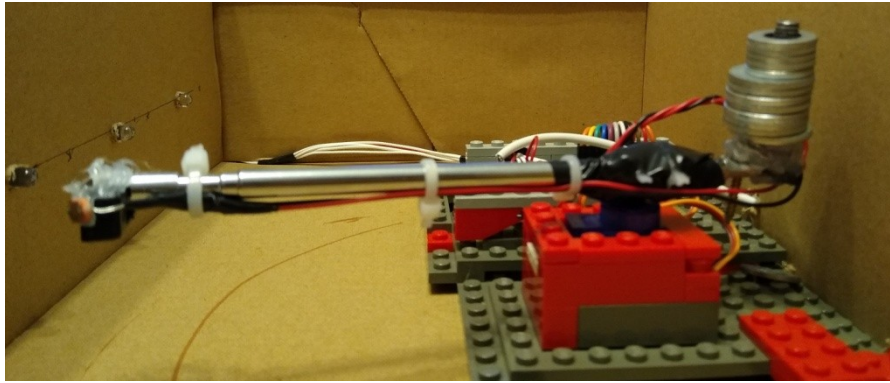
Ze względu na prototypowy charakter projektu rozwiązania mechaniczne nie są trwałe. Wykorzystane zostały powszechnie dostępne materiały. Konstrukcję przedstawia Rysunek 1.



Rysunek 1 Konstrukcja układu

W zamkniętej obudowie wykonanej z kartonu, na jednej ze ścianek umieszczono w równej odległości od siebie i na tej samej wysokości pięć punktowych źródeł światła – białych diod LED. Po przeciwległej stronie obudowy umieszczono manipulator.

Manipulator składa się z serwomechanizmu umieszczonego w stabilnej podstawie, do którego przymocowane zostało metalowe ramię, posiadające na dalszym końcu dwa fotorezystory, pełniące w układzie rolę czujników natężenia światła. Postawa manipulatora wykonana została z klocków LEGO, a samo ramię manipulatora – z części pozyskanych z odzysku ze starych urządzeń. Jako napęd użyty został serwomechanizm Micro SG90. Dodatkowo wykonana została prowizoryczna przeciwwaga ramienia manipulatora w celu ustabilizowania pracy manipulatora. Zdjęcie poglądowe ramienia ukazuje Rysunek .



Rysunek 2 Ramię manipulatora

2.2 UKŁAD ELEKTRONICZNY

Układ elektroniczny wykonany został na uniwersalnej płytce prototypowej. Schematy elektroniczne zostały dodane jako załącznik do niniejszego dokumentu. Układ jest zasilany uniwersalną ładowarką sieciową o napięciu znamionowym 5V DC.

3. STEROWANIE URZĄDZENIAMI WYKONAWCZYM

3.1 SPECYFIKACJA

Funkcjonalność układu zakłada sterowanie pięcioma diodami LED oraz serwomechanizmem. Układ ma umożliwić sterowanie każdej z diod osobno (ustawianie stanów wysokich oraz niskich), oraz ustawianie serwomechanizmu w pozycji minimalnej (0°) oraz maksymalnej (180°).

3.2 IMPLEMENTACJA STEROWANIA DIODAMI LED

Skonfigurowano pięć wyjść. Opis trybów pracy wyjść cyfrowych oraz piny, na których zostały skonfigurowane wyszczególniono w podpunkcie 8.1 Raport programu STM32CubeMX z opisem wyprowadzeń i konfiguracji mikrokontrolera „PID Solar Tracker Project - Configuration Report“. Implementacja programowa sterowania diodami została umieszczona w funkcji *SetDevice*, która znajduje się w pliku *functions.c*. Do wystrojenia diod została wykorzystana wbudowana funkcja biblioteki HAL – *HAL_GPIO_WritePin*. Definicje funkcji *SetDevice* przedstawia Listing 2.

3.3 IMPLEMENTACJA STEROWANIA SERWOMECHANIZMEM

Skonfigurowano *Timer 9* z parametrami określonymi w punkcie 7.9 Raport programu STM32CubeMX z opisem wyprowadzeń i konfiguracji mikrokontrolera „PID Solar Tracker Project - Configuration Report“. Kanał pierwszy tego licznika, został skonfigurowany jako wyjście PWM. Częstotliwość pracy licznika została wyznaczona na 50Hz, ponieważ taka jest optymalna częstotliwość pracy używanego serwomechanizmu. Parametry licznika wyznaczono używając wzoru (1). Numer pinu oraz jego konfiguracja została opisana w punkcie 8.1 Raport programu STM32CubeMX z opisem wyprowadzeń i konfiguracji mikrokontrolera „PID Solar Tracker Project - Configuration Report“. Implementacja programowa poruszania serwomechanizmem została opisana w funkcji *MoveServo*. Funkcja ta, przyjmuje jako argument typu *Integer* kąt, na który serwomechanizm ma być ustawiony, następnie oblicza odpowiednią wartość rejestru, do którego ma wpisać wypełnienie PWM kanału pierwszego. Do wpisania do rejestru informacji używa wbudowanego makra biblioteki HAL:

`__HAL_TIM_SET_COMPARE`. Definicje funkcji *MoveServo* przedstawia Listing 1. Do przeliczenia zadanego kąta na wartość rejestru *CCR1 timera*, zostały użyte zmienne *SERVO_PARAMETER* oraz *SERVO_CONST*, które zostały wyznaczone sposobem eksperymentalnym. Zgodnie z notą katalogową (Dokumentacja serwo mechanizmu „SERVO MOTOR SG90 Datasheet”, serwo mechanizm sterowany jest wypełnieniem PWM od 5% dla zerowego położenia, oraz 20% wypełnienia PWM dla maksymalnego położenia.

$$f_{TIMER} = \frac{f_{CLK}}{(COUNTER + 1)(PRESCALER + 1)(CLKDIV) + 1} \quad (1)$$

Gdzie:

- f_{TIMER} - częstotliwość timera (częstotliwość przzerwania od upływu okresu *timera*)
- f_{CLK} – częstotliwość sygnału zegarowego magistrali wybranego *timera*.
- COUNTER - wartość okresu licznika,
- PRESCALER - wartość *preskalera*,
- CLKDIV - wartość wewnętrznego dzielnika zegara

Listing 1 Implementacja funkcji *MoveServo*

```
1 void MoveServo(int deg)
2 {
3     uint16_t SET_COMPARE = deg * SERVO_PARAMETER + SERVO_CONST;
4     __HAL_TIM_SET_COMPARE(&htim9, TIM_CHANNEL_1, SET_COMPARE);
5 }
```

3.4 IMPLEMENTACJA STEROWANIA URZĄDZEŃ WYJŚCIOWYCH

Do sterowania urządzeniami wyjściowymi, została utworzona funkcja *SetDevice*, którą przedstawia Listing 2. Funkcja przyjmuje jako argumenty: rodzaj urządzenia (LED/SRV), numer urządzenia (LED: <1:5>, serwo mechanizm: {1}), oraz wartość jaką ma przyjąć (dla diod 0!/0, dla serwo mechanizmu <0:180>). Szczegółowa składnia komendy została opisana w punkcie 5.2. Funkcja zwraca 1, w przypadku, gdy parametry były w określonych odgórnie zakresach, w innym przypadku zwraca 0. Zwracana wartość jest wykorzystywana przez funkcję *SendACK* do wysyłania komunikatów potwierdzenia otrzymania informacji. Procedura UART i potwierdzania informacji opisana jest w dalszych podpunktach.

Listing 2 Implementacja funkcji *SetDevice*

```
1 int SetDevice(char deviceType[], int deviceNumb, int val)
2 {
3     int result = 0;
4     if(!strcmp(deviceType, "LED"))
5     {
6         result = 1;
7         switch(deviceNumb)
8         {
9             case 1: HAL_GPIO_WritePin(USER_LD1_GPIO_Port, USER_LD1_Pin, val) ; break;
10            case 2: HAL_GPIO_WritePin(USER_LD2_GPIO_Port, USER_LD2_Pin, val) ; break;
11            case 3: HAL_GPIO_WritePin(USER_LD3_GPIO_Port, USER_LD3_Pin, val) ; break;
12            case 4: HAL_GPIO_WritePin(USER_LD4_GPIO_Port, USER_LD4_Pin, val) ; break;
13            case 5: HAL_GPIO_WritePin(USER_LD5_GPIO_Port, USER_LD5_Pin, val) ; break;
```

```
14         default:result=0;
15     }
16 }
17 else if(!strcmp(deviceType,"SRV"))
18 {
19     if(val<=180 && val >=0)
20     {
21         MoveServo(val);
22         result=1;
23     }
24 }
25 else if(!strcmp(deviceType,"AUT"))
26 {
27     if(val == 1)
28     {
29         autoMode = 1;
30         result=1;
31     }
32     else if(val ==0) autoMode = 0;
33     result =1;
34 }
35 else result =0;
36 return result;
37 }
```

3.5 WYNIKI TESTÓW

W przypadku poprawnego otrzymania i zdekodowania informacji otrzymanych przez UART, funkcje działały zgodnie z ich przeznaczeniem. W Przypadku, gdy informacje otrzymane przez UART były nieprawidłowe (np. zadany kąt obrotu serwomechanizmu 300°) funkcja ze względu na rozpisane instrukcje warunkowe, nie sterowała urządzeniami wyjściowymi.

3.6 PODSUMOWANIE

Funkcje przedstawione na listingach: Listing 1 oraz Listing 2 oraz prawidłowa konfiguracja peryferii umożliwiają zdalne sterowanie urządzeniami wyjściowymi. Diody 1-5 gasną i zapalają się, a serwomechanizm porusza się na zadany przez użytkownika kąt, o ile mieści się w założonym ogólnie zakresie. Świadczy to o poprawnymysterowaniu serwomechanizmu sposobem eksperymentalnym.

4. POMIAR NATĘŻENIA ŚWIATŁA

4.1 SPECYFIKACJA

Funkcjonalność zakłada odczyt z dwóch czujników natężenia światła. Informacja jaka jest odczytywana, to sygnał napięciowy <0;3300> mV.

4.2 IMPLEMENTACJA

Skonfigurowano dwa przetworniki analogowo-cyfrowe: ADC1, ADC2. Opis trybów pracy wejść analogowych oraz piny, na których zostały skonfigurowane wyszczególniono w podpunkcie 8.1 Raport programu STM32CubeMX z opisem wyprowadzeń i konfiguracji mikrokontrolera „PID Solar Tracker Project - Configuration Report“. Implementacja programowa sterowania diodami została umieszczona w funkcji *ReadSensors*, która znajduje się

na Listing 3. Do pobrania aktualnych wartości czujników, użyto funkcji zdefiniowanych w bibliotece HAL: *HAL_ADC_GetValue* oraz *HAL_ADC_PoolForConversion*. Funkcja *ReadSensors* odczytuje wartości sygnałów napięciowych z czujników 1 oraz 2. Wartości te przechowywane są w zmiennych globalnych *lightSensor1* i *lightSensor2*. Funkcja zwraca średnią wartość odczytanych z czujników. Pętla znajdująca się z linii 8 i 9, oczekuje, aż odczyt z czujników zostanie zakończony.

Listing 3 Implementacja funkcji *ReadSensors*.

```
1 uint32_t ReadSensors()
2 {
3
4     HAL_ADC_Start(&hadc1);
5     HAL_ADC_Start(&hadc2);
6
7     /* Wait until readings are done */
8     while((HAL_ADC_PollForConversion(&hadc1, 10) != HAL_OK) &&
9           (HAL_ADC_PollForConversion(&hadc2, 10) != HAL_OK));
10
11     lightSensor1 = (uint32_t) (HAL_ADC_GetValue(&hadc1) / ((float) 0xffff) * MAX_VOLTAGE*1000);
12     lightSensor2 = (uint32_t) (HAL_ADC_GetValue(&hadc2) / ((float) 0xffff) * MAX_VOLTAGE*1000);
13     uint32_t result = ((lightSensor1 + lightSensor2))/2;
14     return result;
15 }
```

4.3 WYNIKI TESTÓW

Po wywołaniu funkcji do zmiennych *lightSensor1* *lightSensor2* przypisywane są wartości aktualnych odczytów z czujników podłączonych do wejść ADC1 i ADC2. Funkcja zwraca uśrednioną wartość obu czujników, a więc zachowuje się zgodnie z założeniami programisty.

4.4 PODSUMOWANIE

Zaimplementowane rozwiązanie pozwala na odczyt wartości czujników. Wartości te, sprawdzane są cyklicznie w funkcji *HAL_TIM_PWM_PulseFinishedCallback*, czyli przy zakończonym pulsie *Timera* 9, aby przechowywać aktualne odczyty.

5. OBSŁUGA KOMUNIKACJI SZEREGOWEJ WRAZ Z KONTROLĄ BŁĘDÓW

5.1 SPECYFIKACJA

Funkcjonalność zakłada odbieranie komunikatów służących odpowiedniemuysterowaniu urządzeń, wysyłanie potwierdzeń po wykonaniu żądanych zmian oraz wysyłanie bieżących odczytów z czujników. Transmisja szeregowo została uruchomiona na porcie 3. Dane konfiguracyjne znajdują się w punkcie 7.9 Raport programu STM32CubeMX z opisem wyprowadzeń i konfiguracji mikrokontrolera „PID Solar Tracker Project - Configuration Report”.

5.2 IMPLEMENTACJA

Utworzono bufor typu *Char* na dane wejściowe o rozmiarze 30 znaków. Ustalono konkretną postać komunikatu, która powinna zostać odebrana („*DEV x=yyy\r*”) gdzie:

- *DEV* – rodzaj urządzenia, dla którego powinna zostać uruchomiona obsługa (SRV dla serwomechanizmu, LED dla diody, AUT dla zmiany trybu AUTO/MANUAL o czym więcej w punkcie 5)
- *x* – Numer urządzenia (Serwomechanizm – 1, tryb Auto – 1, Led : <1:5>)
- *yyy* – Wartość do nadania (Led {0;!0}, Serwomechanizm: <0:180>, zmiana Trybu: {0,!0})
- *\r* – znak końca linii (HEX = 0x0D, ASCII = 13)

Po otrzymaniu znaku, wywoływana jest funkcja *HAL_UART_RxCpltCallback*, w której sprawdzane jest, czy otrzymany znak jest znakiem końca linii. Jeżeli tak, to sprawdzana jest również długość otrzymanego ciągu znaków (zastosowano mechanizm inkrementacji indeksu wstawianego znaku do bufora) ponieważ wszystkie komunikaty powinny być takiej samej długości. W przypadku, gdy znak końca linii ma numer indeksu inny od oczekiwanego, wywoływana jest procedura obsługi błędu, wywoływana jest funkcja *LossOfData*, która jako argumenty przyjmuje bufor, jego rozmiar oraz handler UART. Funkcja ta, wywołuje funkcję czyszczącą bufor – *ClearBuffer*, a następnie wysyła komunikat o tym, że dane zostały utracone w trakcie transmisji. W przypadku, gdy komunikat jest prawidłowy, wysyłane jest potwierdzenie otrzymania prawidłowego komunikatu i wywoływana jest funkcja *SendACK* opisywana w punkcie 2. Definicja funkcji *HAL_UART_RxCpltCallback* została zamieszczona na listingu 4, definicje funkcji *LossOfData*, oraz *ClearBuffer* zostały przedstawione kolejno na Listing 5 i Listing 6.

Listing 4 Definicja funkcji *LossOfData*

```
1 void LossOfData(char buffer[], int buffer_size, UART_HandleTypeDef *huart)
2 {
3     char message[] = "Loss Of Data\r";
4     ClearBuffer(buffer, buffer_size);
5     HAL_UART_Transmit_IT(huart, (uint8_t*)message, (sizeof(message)/sizeof(char))-1);
6 }
```

Listing 5 Definicja funkcji *ClearBuffer*

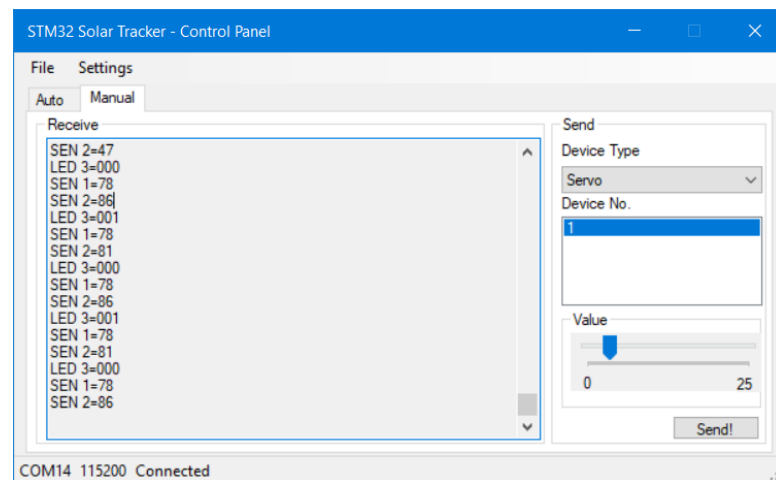
```
1 void ClearBuffer(char buff[], int size)
2 {
3     while(size != 0)
4     {
5         buff[size]=0;
6         size--;
7     }
8 }
```

Listing 6 Ciało funkcji HAL_UART_RxCpltCallback

```
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
1 {
2     if(huart->Instance == USART3)
3     {
4         datalost=0;
5         if(received_message[idx] == 0x0D)
6         {
7
8             if(idx == 9)
9             {
10                sscanf(received_message, "%3s %d=%d", &deviceType, &device_number, &value);
11                if(SetDevice(deviceType, device_number, value) == 1)
12                {
13                    SendACK(deviceType, device_number, value);
14                    ClearBuffer(received_message, MESSAGE_SIZE);
15                    idx=0;
16                    datalost = 1;
17                }
18            }
19        }
20        else if(idx > 9) datalost = 0;
21        else
22        {
23            idx++;
24            datalost = 1;
25        }
26
27        if(datalost==0)
28        {
29            idx=0;
30            LossOfData(received_message, MESSAGE_SIZE, huart);
31            datalost = 1;
32        }
33
34    }
35    HAL_UART_Receive_IT(&huart3, &received_message[idx], 1);
36 }
37
```

5.3 WYNIKI TESTÓW

Po naciśnięciu przycisku na port szeregowy wysłany został odczyt pomiaru, co można zauważyć na ostatnich dwóch liniach pola *Receive* na Rysunek :



Rysunek 3 Widok dedykowanego programu w trybie ręcznym – odczyty z portu szeregowego

5.4 PODSUMOWANIE

Zaimplementowane rozwiązanie pozwala na odbieranie komunikatów wraz z obsługą błędów oraz wysyłanie potwierdzeń ich otrzymania. Zastosowany algorytm komunikacji umożliwia bezawaryjne funkcjonowanie układu sterowania.

6. ALGORYTM STEROWANIA

6.1 SPECYFIKACJA

Funkcjonalność zakłada ustawianie ramienia manipulatora w kierunku najjaśniejszego źródła światła. Po wyzwoleniu przyciskiem *USER_BTN1* urządzenie wykonuje skanowanie: sprawdza odczyty dla każdej z pozycji zmieniając położenie o jeden stopień co sto milisekund, a następnie ustawia ramię w optymalnej pozycji. Po pierwszym skanowaniu manipulator przechodzi w tryb poprawiania pozycji, w którym to w przypadku wykrycia zmiany odczytów większej niż założona, poprawia swoją pozycję w czasie rzeczywistym.

6.2 IMPLEMENTACJA

W pętli głównej umieszczona jest obsługa przycisku *USER_BTN1* w trybie *poolingu*. Naciśnięcie tego przycisku wywołuje funkcję *ScanArea*, która jako argument przyjmuje wskaźnik na 32-bitową zmienną typu *Unsigned Integer* – *maxResult*. Zmienna ta przechowuje maksymalny uśredniony odczyt czujników. Ponadto, funkcja zwraca wartość kąta, dla którego znalazła maksymalny odczyt (typu *Integer*). Po zakończeniu skanowania, ramię manipulatora ustawiane jest w pozycji, dla której odczyt z czujników był największy. Po zakończeniu skanowania, uruchamiany jest tryb kalibracji pozycji. Przy każdym odczytaniu wartości czujników (wywoływany w *HAL_TIM_PWM_PulseFinishedCallback*), aktualny stan porównywany jest z tym zwróconym z funkcji *ScanArea*. Jeżeli odczyt różni się w sposób przekraczający granicę błędu wyznaczoną podczas eksperymentów, pozycja ramienia zostanie skorygowana (wywołanie funkcji *RecalibratePosition*), a wartości zmiennych *actualAngle* i *actualValue* przechowujących informację o aktualnym kącie i wartości z czujników zostaną nadpisane.

Po przejściu w zakładkę *Manual* w dedykowanej aplikacji zewnętrznej, wysyłany jest sygnał „AUT 1=00x\r lub AUT 1=00x\r”. Po odebraniu takiego sygnału, tryb kalibracji zostaje tymczasowo wyłączony ($x=0$) i umożliwiające zostaje manualne ustawianie pozycji ramienia manipulatora. Po przejściu w zakładkę *Auto*, wysłany zostaje odpowiedni komunikat (z $x!=0$), po którym tryb kalibracji pozycji zostaje wznowiony. Definicję funkcji *ScanArea* przedstawiono na Listing 7, definicję funkcji *RecalibratePosition* na Listing 8, a ciało funkcji *HAL_TIM_PWM_PulseFinishedCallback* na Listing 9

Listing 7 Definicja funkcji *ScanArea*

```
1 int ScanArea(uint32_t *maxResult)
2 {
3     scanFlag=1;
4     uint32_t dataBuffer[180] = {};
5
6     int resultAngle=0; //wyni kąta
7     uint32_t max_result=0; //?
8     for(int angle=0; angle<180;angle++)
9     {
10         MoveServo(angle);
11         HAL_Delay(100);
12         dataBuffer[angle] = ReadSensors();
13         if(dataBuffer[angle] > max_result)
14         {
15             max_result = dataBuffer[angle];
16             resultAngle = angle;
17         }
18         //Send information about actual position
19         SendACK("SRV", 1,angle);
20     }
21     ///SendACK("SRV", 1, resultAngle);
22     MoveServo(resultAngle);
23     SendACK("SRV", 1,resultAngle);
24
25     //return maxResult via reference
26     //if(mode == 0) HAL_TIM_Base_Start_IT(&htim7);
27     actualValue = max_result;
28     scanFlag = 0;
29     return resultAngle;
30 }
```

Listing 8 Definicja funkcji *RecalibratePosition*

```
1 int RecalibratePosition()
2 {
3
4     if(lightSensor1 > 500 || lightSensor2 >500)
5     {
6         if(lightSensor2 > lightSensor1)
7         {
8             if(actualAngle > 0)
9             {
10                 actualAngle--;
11             }
12         }
13         else
14         {
```

```
15         if(actualAngle < 180)
16         {
17             actualAngle++;
18         }
19
20     }
21 }
22 MoveServo(actualAngle);
23 SendACK("SRV", 1, actualAngle);
24 return actualAngle;
25 }
```

Listing 9 Ciało funkcji HAL_TIM_PWM_PulseFinishedCallback

```
void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *htim)
1 {
2     if(htim->Instance == TIM9 && scanFlag == 0 && autoMode == 1)
3     {
4         /*          Get new reading from sensors every pulse, if mean
5 value of sensors readings differs
6          *          more than specified, recalibrate servo position
7          */
8         temp_val = ReadSensors();
9         if(abs(actualValue - temp_val)/40 > 10 && (actualValue != 0)
10 && (scanFlag == 0))
11         {
12             actualAngle = RecalibratePosition();
13             actualValue = temp_val;
14         }
15     }
16 }
```

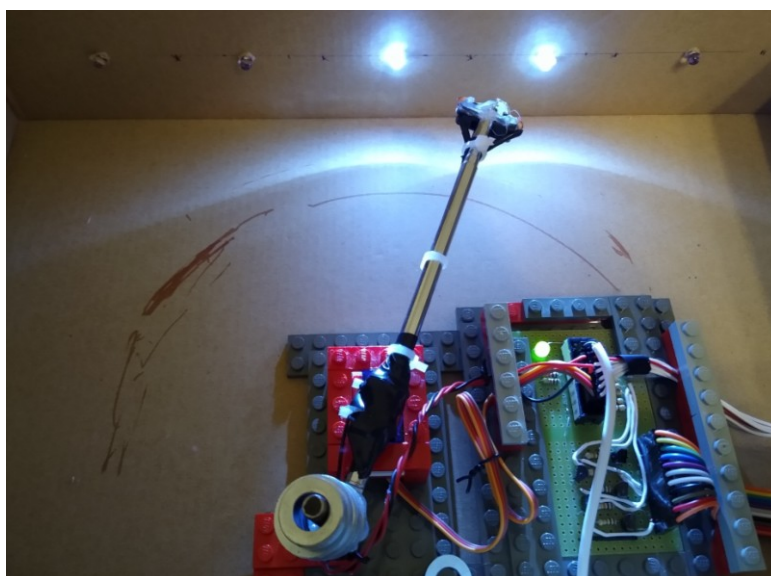
6.3 WYNIKI TESTÓW

Działanie układu przedstawione zostało podczas prezentacji projektu. Ponadto, zdjęcia układu podczas pracy zamieszczono na poniższych Rysunkach. Rysunek 4 przedstawia sytuację, gdy zapalona została skrajna prawa dioda (numer 5), następnie naciśnięto przycisk *USR_BTN1*. Naciśnięcie wywołało skanowanie obszaru (funkcja *ScanArea*), po czym ramie manipulatora ustawiło się w pozycji, w której odczyt z czujników był największy.

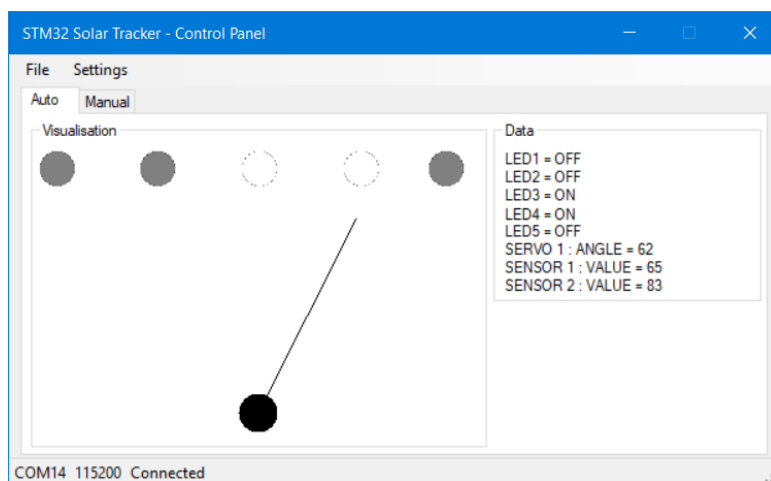
Rysunek przedstawia sytuację, gdy po ustawieniu się ramienia na opisanej wyżej pozycji, zmieniono stan zapalonych diod LED. Manipulator po wykryciu zmiany w odczytach czujników, dostosował swoją pozycję. Poprawne działanie układu potwierdza wizualizacja stworzona w dedykowanej aplikacji okienkowej, która opiera się na potwierdzeniach opisywanych szerzej w punkcie 5.2. Wizualizacja przedstawiona została na Rysunek .



Rysunek 4 Wynik działania funkcji ScanArea



Rysunek 5 Dostrojenie pozycji manipulatora po zmianie stanu diod LED



Rysunek 6 Odzwierciedlenie faktycznej pozycji manipulatora na wizualizacji

6.4 PODSUMOWANIE

Regulacja pozycji w zależności od odczytów czujników, zmieniających się wraz z natężeniem oświetlenia została przeprowadzona pomyślnie. Pokazane zostało to w powyższym podpunkcie, oraz podczas prezentacji działania układu.

7. DEDYKOWANY PROGRAM OKIENKOWY

7.1 SPECYFIKACJA

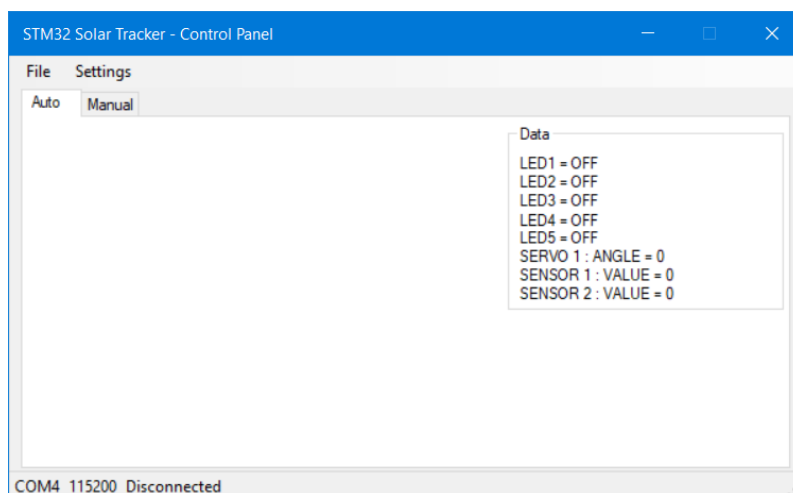
Dedykowany program okienkowy ma za zadanie obsługiwać komunikację szeregową między mikrokontrolerem a komputerem osobistym. Funkcjonalność programu zakłada:

- możliwość połączenia się za pomocą komunikacji szeregowej z mikrokontrolerem, a także zmianę parametrów tej komunikacji (port, prędkość transmisji);
- możliwość odczytu stanu czujników, diod LED oraz serwomechanizmu;
- zmianę stanu ww. urządzeń w tzw. trybie manualnym poprzez wysłanie odpowiednio przygotowanego komunikatu UART;
- graficzną wizualizację stanu diod i położenia manipulatora w tzw. trybie automatycznym;

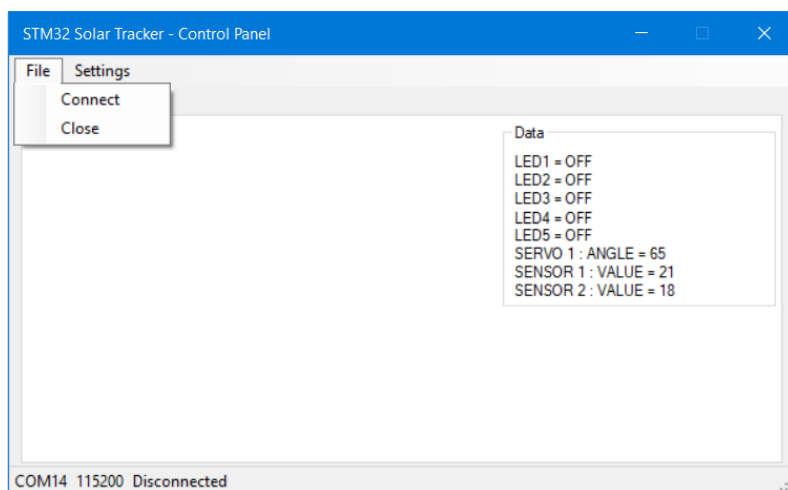
7.2 IMPLEMENTACJA

7.2.1 Interfejs graficzny

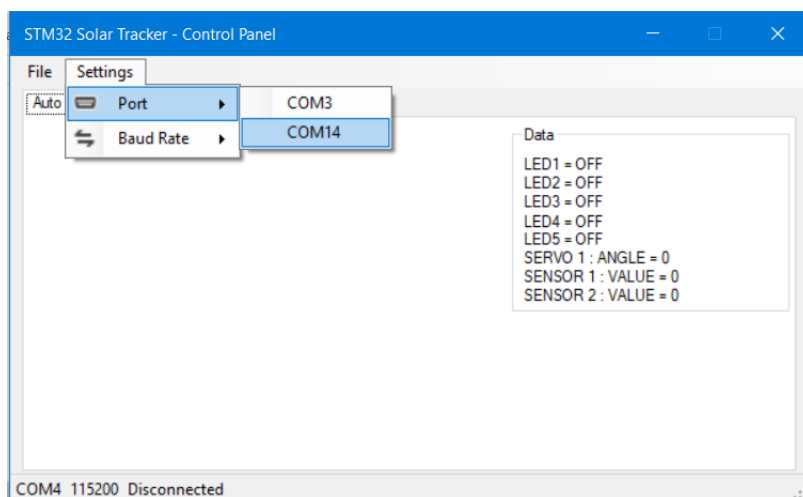
Układ interfejsu graficznego został przedstawiony na poniższych ilustracjach. Rysunek przedstawia widok okna programu po uruchomieniu. Rysunek przedstawia kontrolki menu Plik, natomiast Rysunek oraz Rysunek – kontrolki menu Ustawienia, które umożliwiają ustawienie parametrów komunikacji szeregowej. Funkcjonalności trybu automatycznego przedstawia Rysunek 2, Rysunek 3 i Rysunek 4 prezentują funkcjonalności trybu manualnego oraz pole wyświetlające odebrane komunikaty.



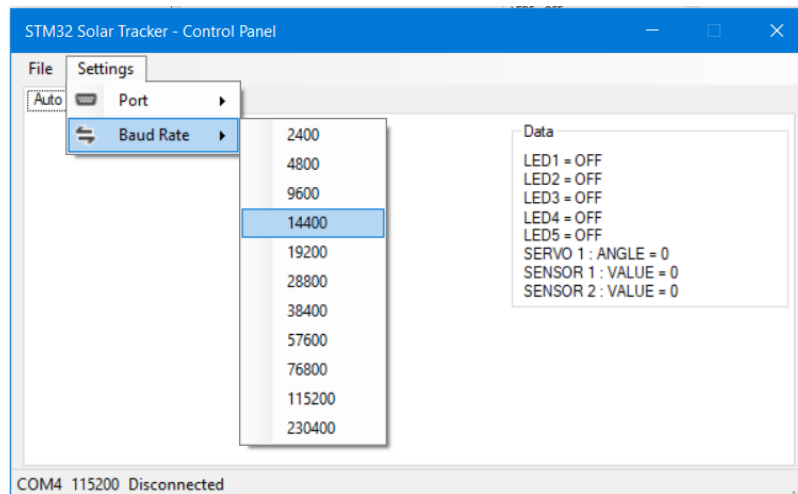
Rysunek 7 Okno programu po uruchomieniu



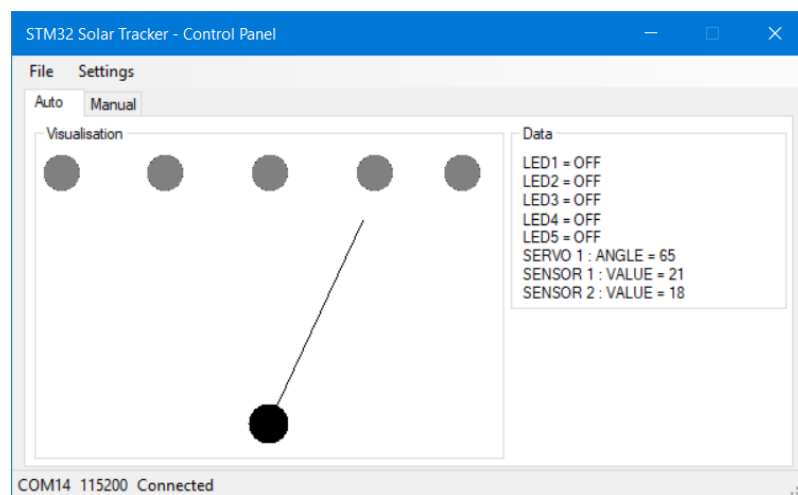
Rysunek 8 Menu plik



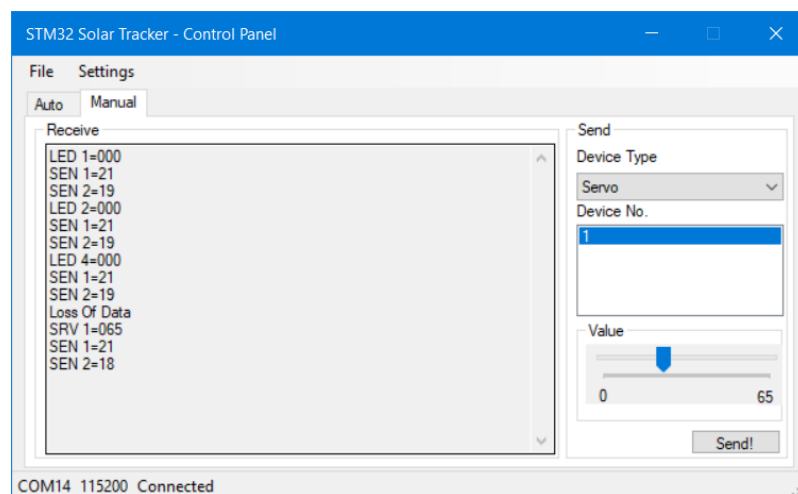
Rysunek 9 Menu wyboru ustawień portu szeregowego



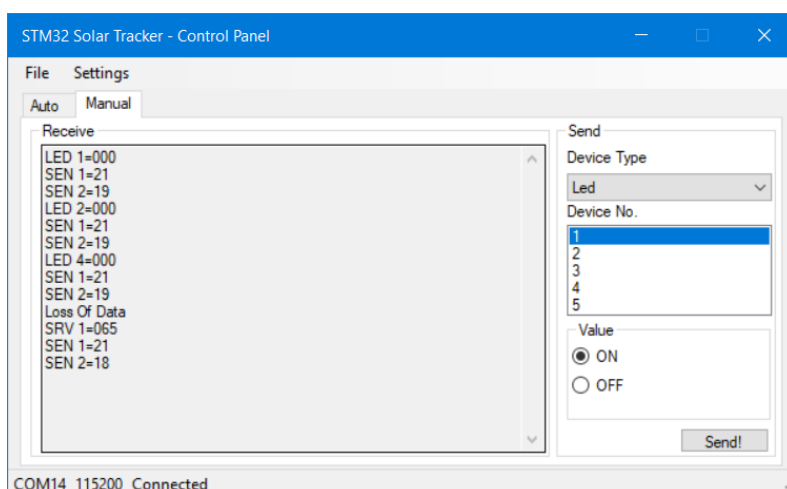
Rysunek 10 Menu ustawień wyboru prędkości transmisji



Rysunek 2 Widok trybu automatycznego



Rysunek 3 Tryb manualny – ustawienia serwomechanizmu



Rysunek 4 Tryb manualny – ustawienia diod LED

6.2.2. Zasada działania

Przedstawiono tutaj jedynie ogólną zasadę działania programu. Kod źródłowy wraz z opisami poszczególnych funkcji stanowi załącznik do niniejszego dokumentu.

Po uruchomieniu wyświetla się okno programu – domyślnie w trybie automatycznym. Jednocześnie wywołana zostaje funkcja *closeConnection*, która zamyka ewentualne otwarte połączenie komunikacji szeregowej. Aby połączyć się z mikrokontrolerem użytkownik musi wybrać port szeregowy oraz prędkość transmisji, a następnie wybrać opcję File > Connect. Wywołana zostaje funkcja *openConnection()*. Jeżeli połączenie zostanie poprawnie nawiązane wyświetli się okno z informacją o udanym uruchomieniu połączenia.

Od tego momentu w oknie trybu automatycznego (zakładka *Auto*) wyświetlać się będzie graficzna wizualizacja stanu układu w czasie rzeczywistym oraz bieżące odczyty wartości sensorów i stanów urządzeń wykonawczych. W tym trybie użytkownik ma możliwość jedynie zmiany stanu diod LED poprzez kliknięcie ich na wizualizacji.

Po przełączeniu na zakładkę *Manual* do mikrokontrolera wysyłana jest informacja o przejściu w tryb manualny. W oknie programu pojawia się pole tekstowe wyświetlające na bieżąco otrzymywane komunikaty. Po prawej stronie pojawia się natomiast pole *Send*, w którym użytkownik ma możliwość zmiany zarówno stanu diod LED jak i zmiany nastawy położenia serwomechanizmu. Po wybraniu żądanych nastaw i kliknięciu przycisku *Send!* wysyłany jest do mikrokontrolera odpowiednio przygotowany komunikat.

Przy żądaniu zamknięcia programu przez użytkownika najpierw zamykane jest połączenie komunikacji szeregowej, a następnie dopiero cały program. Jest to bardzo istotne ze względu na to, że odbiór informacji z portu szeregowego jest realizowany w trybie ciągłym w oddzielnym wątku i zamknięcie programu bez wcześniejszego zakończenia pracy funkcji wywoływanej w osobnym wątku powoduje poważne błędy w działaniu programu.

7.3 WYNIKI TESTÓW

Peryferia układu zostały wysterowane prawidłowo w każdym z wcześniejszych testów, co świadczy o poprawnym działaniu programu. Również tryb automatyczny zadziałał poprawnie – graficzna wizualizacja układu wyświetlała stan zgodny ze stanem rzeczywistym, a odczyty i pozycja serwomechanizmu były aktualizowane na bieżąco.

7.4 PODSUMOWANIE

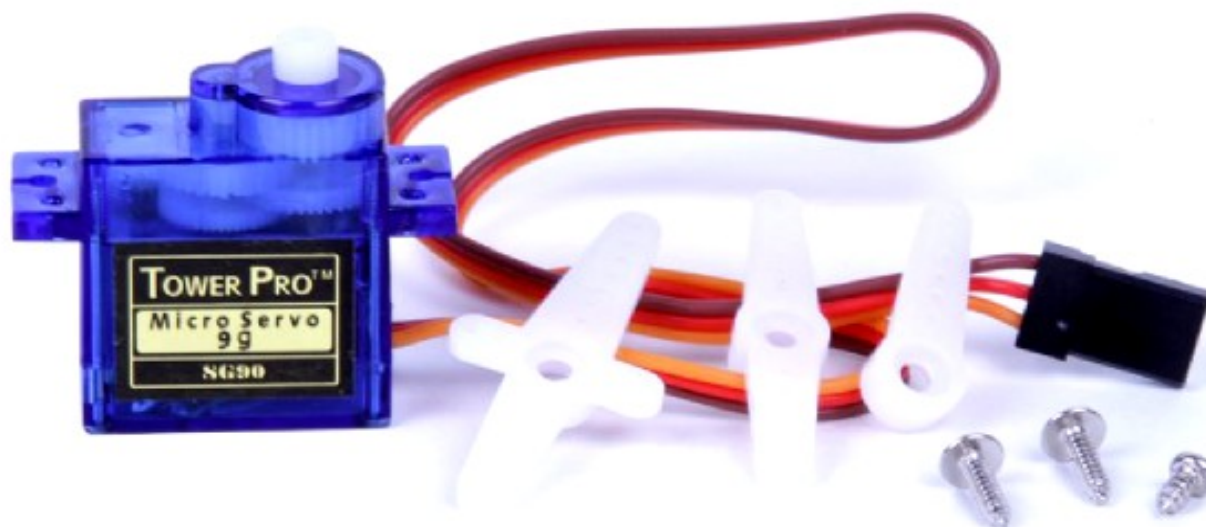
Napisany program umożliwia prostą i intuicyjną obsługę skonstruowanego układu. Ze względu na specyfikę konstrukcji mechanicznej (urządzenie pracuje w zamkniętej obudowie), wizualizacja okazała się przydatnym elementem przy testowaniu działania układu regulacji. Program był również znaczącym ułatwieniem przy testowaniu pozostałych funkcjonalności układu.

8. LITERATURA

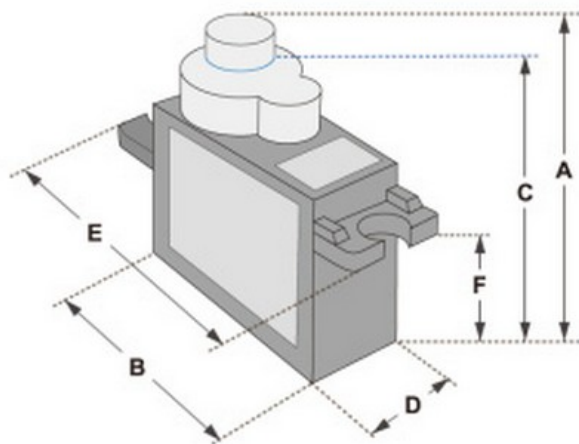
1. DOMINIK ŁUCZAK. Szablon sprawozdania [online]. 2014. s.l.: s.n. Pobrano: <http://zsep.cie.put.poznan.pl/>.
2. MAREK GALEWSKI. STM32 – Aplikacje i ćwiczenia w języku C z biblioteką HAL, Wydawnictwo BTC, Legionowo 2019
3. ADRIAN WÓJCIK. Programowalne układy licznikowe, Materiały do zajęć laboratoryjnych [online]. 2019. Pobrano: <http://zsep.cie.put.poznan.pl/>.

9. ZAŁĄCZNIKI

1. Dokumentacja serwomechanizmu „SERVO MOTOR SG90 Datasheet”
2. Raport programu STM32CubeMX z opisem wyprowadzeń i konfiguracji mikrokontrolera „PID Solar Tracker Project - Configuration Report”
3. Schematy elektryczne układu
4. Repozytorium projektu zawierające kod źródłowy programu mikrokontrolera w witrynie GitHub [online] : <https://github.com/Shakkozu/STM32-Solar-Tracker/tree/master/Inc>
5. Repozytorium projektu zawierające kod źródłowy dedykowanego programu okienkowego w witrynie GitHub [online] : <https://github.com/macieksadowski/STM32-Solar-Tracker-Control-Panel>



Tiny and lightweight with high output power. Servo can rotate approximately 180 degrees (90 in each direction), and works just like the standard kinds but smaller. You can use any servo code, hardware or library to control these servos. Good for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. It comes with a 3 horns (arms) and hardware.

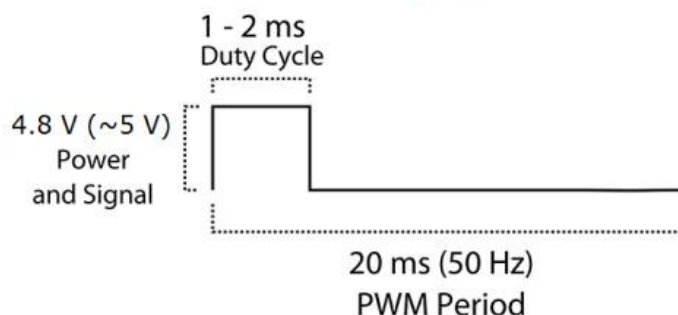


Dimensions & Specifications

A (mm) : 32
B (mm) : 23
C (mm) : 28.5
D (mm) : 12
E (mm) : 32
F (mm) : 19.5
Speed (sec) : 0.1
Torque (kg-cm) : 2.5
Weight (g) : 14.7
Voltage : 4.8 - 6

Position "0" (1.5 ms pulse) is middle, "90" (~2ms pulse) is middle, is all the way to the right, "-90" (~1ms pulse) is all the way to the left.

PWM=Orange (⏏)
Vcc=Red (+)
Ground=Brown (-)



1. Description

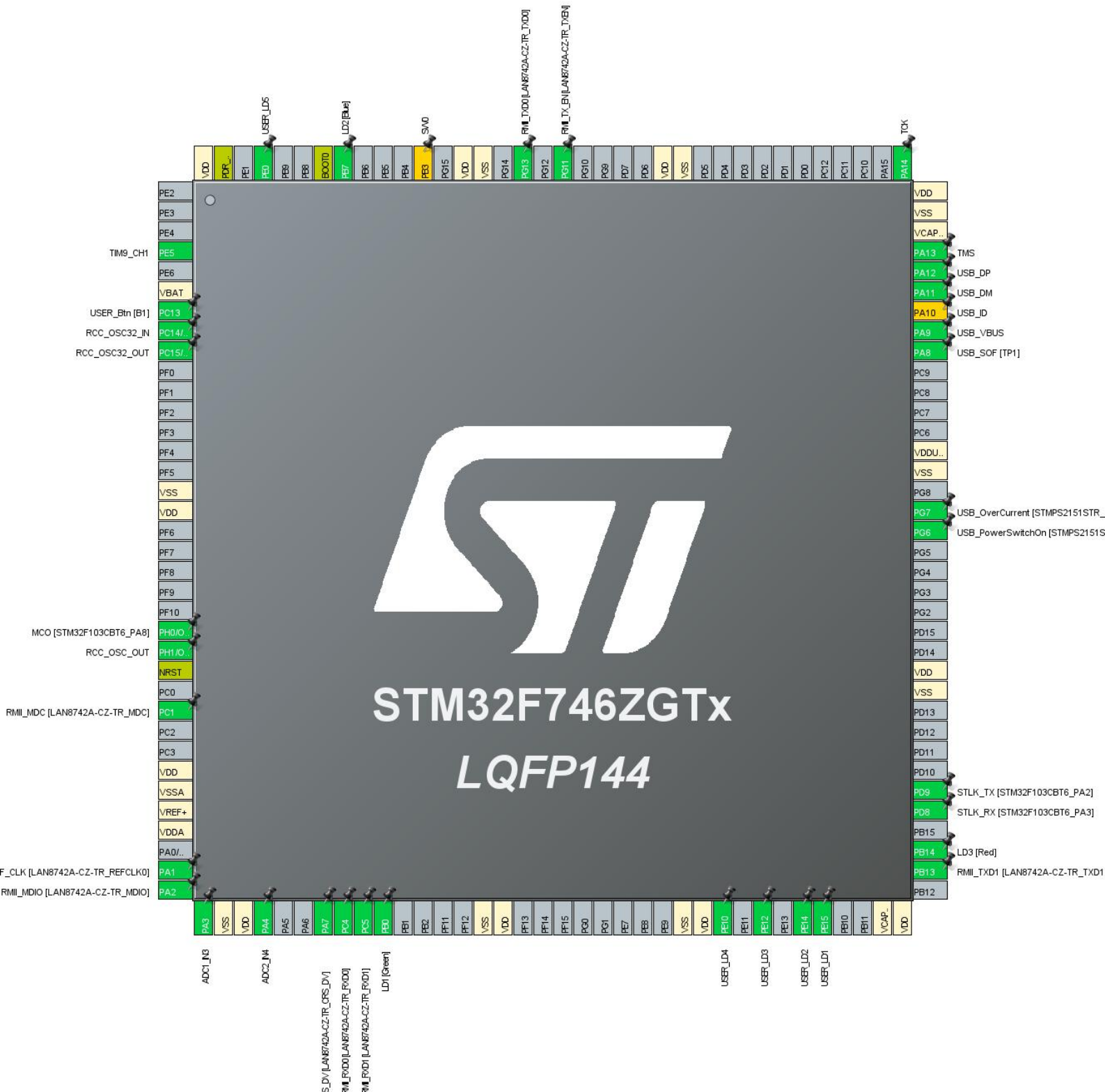
1.1. Project

Project Name	PID Solar Tracker
Board Name	NUCLEO-F746ZG
Generated with:	STM32CubeMX 5.5.0
Date	02/11/2020

1.2. MCU

MCU Series	STM32F7
MCU Line	STM32F7x6
MCU name	STM32F746ZGTx
MCU Package	LQFP144
MCU Pin number	144

2. Pinout Configuration



3. Pins Configuration

Pin Number LQFP144	Pin Name (function after reset)	Pin Type	Alternate Function(s)	Label
4	PE5	I/O	TIM9_CH1	
6	VBAT	Power		
7	PC13	I/O	GPIO_EXTI13	USER_Btn [B1]
8	PC14/OSC32_IN	I/O	RCC_OSC32_IN	
9	PC15/OSC32_OUT	I/O	RCC_OSC32_OUT	
16	VSS	Power		
17	VDD	Power		
23	PH0/OSC_IN	I/O	RCC_OSC_IN	MCO [STM32F103CBT6_PA8]
24	PH1/OSC_OUT	I/O	RCC_OSC_OUT	
25	NRST	Reset		
27	PC1	I/O	ETH_MDC	RMII_MDC [LAN8742A-CZ- TR_MDC]
30	VDD	Power		
31	VSSA	Power		
32	VREF+	Power		
33	VDDA	Power		
35	PA1	I/O	ETH_REF_CLK	RMII_REF_CLK [LAN8742A-CZ- TR_REFCLK0]
36	PA2	I/O	ETH_MDIO	RMII_MDIO [LAN8742A-CZ- TR_MDIO]
37	PA3	I/O	ADC1_IN3	
38	VSS	Power		
39	VDD	Power		
40	PA4	I/O	ADC2_IN4	
43	PA7	I/O	ETH_CRSDV	RMII_CRSDV [LAN8742A- CZ-TR_CRSDV]
44	PC4	I/O	ETH_RXD0	RMII_RXD0 [LAN8742A-CZ- TR_RXD0]
45	PC5	I/O	ETH_RXD1	RMII_RXD1 [LAN8742A-CZ- TR_RXD1]
46	PB0 *	I/O	GPIO_Output	LD1 [Green]
51	VSS	Power		
52	VDD	Power		
61	VSS	Power		
62	VDD	Power		

Pin Number LQFP144	Pin Name (function after reset)	Pin Type	Alternate Function(s)	Label
63	PE10 *	I/O	GPIO_Output	USER_LD4
65	PE12 *	I/O	GPIO_Output	USER_LD3
67	PE14 *	I/O	GPIO_Output	USER_LD2
68	PE15 *	I/O	GPIO_Output	USER_LD1
71	VCAP_1	Power		
72	VDD	Power		
74	PB13	I/O	ETH_TXD1	RMII_TXD1 [LAN8742A-CZ- TR_TXD1]
75	PB14 *	I/O	GPIO_Output	LD3 [Red]
77	PD8	I/O	USART3_TX	STLK_RX [STM32F103CBT6_PA3]
78	PD9	I/O	USART3_RX	STLK_TX [STM32F103CBT6_PA2]
83	VSS	Power		
84	VDD	Power		
91	PG6 *	I/O	GPIO_Output	USB_PowerSwitchOn [STMP2151STR_EN]
92	PG7 *	I/O	GPIO_Input	USB_OverCurrent [STMP2151STR_FAULT]
94	VSS	Power		
95	VDDUSB	Power		
100	PA8	I/O	USB_OTG_FS_SOF	USB_SOF [TP1]
101	PA9	I/O	USB_OTG_FS_VBUS	USB_VBUS
102	PA10 **	I/O	USB_OTG_FS_ID	USB_ID
103	PA11	I/O	USB_OTG_FS_DM	USB_DM
104	PA12	I/O	USB_OTG_FS_DP	USB_DP
105	PA13	I/O	SYS_JTMS-SWDIO	TMS
106	VCAP_2	Power		
107	VSS	Power		
108	VDD	Power		
109	PA14	I/O	SYS_JTCK-SWCLK	TCK
120	VSS	Power		
121	VDD	Power		
126	PG11	I/O	ETH_TX_EN	RMII_TX_EN [LAN8742A- CZ-TR_TXEN]
128	PG13	I/O	ETH_TXD0	RMII_TXD0 [LAN8742A-CZ- TR_TXD0]
130	VSS	Power		
131	VDD	Power		
133	PB3 **	I/O	SYS_JTDO-SWO	SW0
137	PB7 *	I/O	GPIO_Output	LD2 [Blue]

Pin Number LQFP144	Pin Name (function after reset)	Pin Type	Alternate Function(s)	Label
138	BOOT0	Boot		
141	PE0 *	I/O	GPIO_Output	USER_LD5
143	PDR_ON	Reset		
144	VDD	Power		

* The pin is affected with an I/O function

** The pin is affected with a peripheral function but no peripheral mode is activated

4. Clock Tree Configuration

5. Software Project

5.1. Project Settings

Name	Value
Project Name	PID Solar Tracker
Project Folder	C:\Users\user\STM32CubeIDE\workspace_1.0.2\PID Solar Tracker
Toolchain / IDE	STM32CubeIDE
Firmware Package Name and Version	STM32Cube FW_F7 V1.15.0

5.2. Code Generation Settings

Name	Value
STM32Cube MCU packages and embedded software	Copy only the necessary library files
Generate peripheral initialization as a pair of '.c/.h' files	No
Backup previously generated files when re-generating	No
Delete previously generated files when not re-generated	Yes
Set all free pins as analog (to optimize the power consumption)	No

6. Power Consumption Calculator report

6.1. Microcontroller Selection

Series	STM32F7
Line	STM32F7x6
MCU	STM32F746ZGTx
Datasheet	027590_Rev4

6.2. Parameter Selection

Temperature	25
Vdd	3.6

7. IPs and Middleware Configuration

7.1. ADC1

mode: IN3

7.1.1. Parameter Settings:

ADCs_Common_Settings:

Mode Independent mode

ADC_Settings:

Clock Prescaler PCLK2 divided by 2

Resolution 12 bits (15 ADC Clock cycles)

Data Alignment Right alignment

Scan Conversion Mode Disabled

Continuous Conversion Mode **Enabled ***

Discontinuous Conversion Mode Disabled

DMA Continuous Requests Disabled

End Of Conversion Selection EOC flag at the end of single channel conversion

ADC_Regular_ConversionMode:

Number Of Conversion 1

External Trigger Conversion Source Regular Conversion launched by software

External Trigger Conversion Edge None

Rank 1

Channel Channel 3

Sampling Time 3 Cycles

ADC_Injected_ConversionMode:

Number Of Conversions 0

WatchDog:

Enable Analog WatchDog Mode false

7.2. ADC2

mode: IN4

7.2.1. Parameter Settings:

ADCs_Common_Settings:

Mode Independent mode

ADC_Settings:

Clock Prescaler PCLK2 divided by 2

Resolution 12 bits (15 ADC Clock cycles)

Data Alignment	Right alignment
Scan Conversion Mode	Disabled
Continuous Conversion Mode	Enabled *
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Disabled
End Of Conversion Selection	EOC flag at the end of single channel conversion

ADC_Regular_ConversionMode:

Number Of Conversion	1
External Trigger Conversion Source	Regular Conversion launched by software
External Trigger Conversion Edge	None
Rank	1
Channel	Channel 4
Sampling Time	3 Cycles

ADC_Injected_ConversionMode:

Number Of Conversions	0
-----------------------	---

WatchDog:

Enable Analog WatchDog Mode	false
-----------------------------	-------

7.3. CORTEX_M7

7.3.1. Parameter Settings:

Cortex Interface Settings:

Flash Interface	AXI Interface
ART ACCELERATOR	Disabled
Instruction Prefetch	Disabled
CPU ICache	Disabled
CPU DCache	Disabled

Cortex Memory Protection Unit Control Settings:

MPU Control Mode	MPU NOT USED
------------------	--------------

7.4. ETH

Mode: RMII

7.4.1. Parameter Settings:

Advanced : Ethernet Media Configuration:

Auto Negotiation	Enabled
------------------	---------

General : Ethernet Configuration:

Ethernet MAC Address	00:80:E1:00:00:00
PHY Address	0 *

Ethernet Basic Configuration:

Rx Mode	Polling Mode
TX IP Header Checksum Computation	By hardware

7.4.2. Advanced Parameters:

External PHY Configuration:

PHY	LAN8742A_PHY_ADDRESS
PHY Address Value	0
PHY Reset delay these values are based on a 1 ms Systick interrupt	0x000000FF *
PHY Configuration delay	0x00000FFF *
PHY Read TimeOut	0x0000FFFF *
PHY Write TimeOut	0x0000FFFF *

Common : External PHY Configuration:

Transceiver Basic Control Register	0x00 *
Transceiver Basic Status Register	0x01 *
PHY Reset	0x8000 *
Select loop-back mode	0x4000 *
Set the full-duplex mode at 100 Mb/s	0x2100 *
Set the half-duplex mode at 100 Mb/s	0x2000 *
Set the full-duplex mode at 10 Mb/s	0x0100 *
Set the half-duplex mode at 10 Mb/s	0x0000 *
Enable auto-negotiation function	0x1000 *
Restart auto-negotiation function	0x0200 *
Select the power down mode	0x0800 *
Isolate PHY from MII	0x0400 *
Auto-Negotiation process completed	0x0020 *
Valid link established	0x0004 *
Jabber condition detected	0x0002 *

Extended : External PHY Configuration:

PHY special control/status register Offset	0x1F *
PHY Speed mask	0x0004 *
PHY Duplex mask	0x0010 *
PHY Interrupt Source Flag register Offset	0x001D *

PHY Link down interrupt

0x000B *

7.5. GPIO

7.6. RCC

High Speed Clock (HSE): BYPASS Clock Source

Low Speed Clock (LSE) : Crystal/Ceramic Resonator

7.6.1. Parameter Settings:

System Parameters:

VDD voltage (V)	3.3
Flash Latency(WS)	2 WS (3 CPU cycle)

RCC Parameters:

HSI Calibration Value	16
TIM Prescaler Selection	Disabled
HSE Startup Timeout Value (ms)	100
LSE Startup Timeout Value (ms)	5000

Power Parameters:

Power Over Drive	Disabled
Power Regulator Voltage Scale	Power Regulator Voltage Scale 3

7.7. SYS

Debug: Serial Wire

Timebase Source: SysTick

7.8. TIM7

mode: Activated

7.8.1. Parameter Settings:

Counter Settings:

Prescaler (PSC - 16 bits value)	8999 *
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	7999 *
auto-reload preload	Enable *

Trigger Output (TRGO) Parameters:

Trigger Event Selection

Reset (UG bit from TIMx_EGR)

7.9. TIM9

Channel1: PWM Generation CH1

7.9.1. Parameter Settings:

Counter Settings:

Prescaler (PSC - 16 bits value)	11 *
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	59999 *
Internal Clock Division (CKD)	No Division
auto-reload preload	Enable *

PWM Generation Channel 1:

Mode	PWM mode 1
Pulse (16 bits value)	500 *
Output compare preload	Enable
Fast Mode	Enable *
CH Polarity	High

7.10. USART3

Mode: Asynchronous

7.10.1. Parameter Settings:

Basic Parameters:

Baud Rate	115200
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

Advanced Parameters:

Data Direction	Receive and Transmit
Over Sampling	16 Samples
Single Sample	Disable

Advanced Features:

Auto Baudrate	Disable
TX Pin Active Level Inversion	Disable
RX Pin Active Level Inversion	Disable

Data Inversion	Disable
TX and RX Pins Swapping	Disable
Overrun	Enable
DMA on RX Error	Enable
MSB First	Disable

7.11. USB_OTG_FS

Mode: Device_Only

mode: Activate_VBUS

mode: Activate_SOF

7.11.1. Parameter Settings:

Speed	Full Speed 12MBit/s
Low power	Disabled
Link Power Management	Disabled
VBUS sensing	Enabled
Signal start of frame	Enabled

*** User modified value**

8. System Configuration

8.1. GPIO configuration

IP	Pin	Signal	GPIO mode	GPIO pull/up pull down	Max Speed	User Label
ADC1	PA3	ADC1_IN3	Analog mode	No pull-up and no pull-down	n/a	
ADC2	PA4	ADC2_IN4	Analog mode	No pull-up and no pull-down	n/a	
ETH	PC1	ETH_MDC	Alternate Function Push Pull	No pull-up and no pull-down	Very High *	RMII_MDC [LAN8742A-CZ-TR_MDC]
	PA1	ETH_REF_CLK	Alternate Function Push Pull	No pull-up and no pull-down	Very High *	RMII_REF_CLK [LAN8742A-CZ-TR_REFCLK0]
	PA2	ETH_MDIO	Alternate Function Push Pull	No pull-up and no pull-down	Very High *	RMII_MDIO [LAN8742A-CZ-TR_MDIO]
	PA7	ETH_CRS_DV	Alternate Function Push Pull	No pull-up and no pull-down	Very High *	RMII_CRS_DV [LAN8742A-CZ-TR_CRS_DV]
	PC4	ETH_RXD0	Alternate Function Push Pull	No pull-up and no pull-down	Very High *	RMII_RXD0 [LAN8742A-CZ-TR_RXD0]
	PC5	ETH_RXD1	Alternate Function Push Pull	No pull-up and no pull-down	Very High *	RMII_RXD1 [LAN8742A-CZ-TR_RXD1]
	PB13	ETH_TXD1	Alternate Function Push Pull	No pull-up and no pull-down	Very High *	RMII_TXD1 [LAN8742A-CZ-TR_TXD1]
	PG11	ETH_TX_EN	Alternate Function Push Pull	No pull-up and no pull-down	Very High *	RMII_TX_EN [LAN8742A-CZ-TR_TXEN]
	PG13	ETH_TXD0	Alternate Function Push Pull	No pull-up and no pull-down	Very High *	RMII_TXD0 [LAN8742A-CZ-TR_TXD0]
RCC	PC14/OSC32_IN	RCC_OSC32_IN	n/a	n/a	n/a	
	PC15/OSC32_OUT	RCC_OSC32_OUT	n/a	n/a	n/a	
	PH0/OSC_IN	RCC_OSC_IN	n/a	n/a	n/a	MCO [STM32F103CBT6_PA8]
	PH1/OSC_OUT	RCC_OSC_OUT	n/a	n/a	n/a	
SYS	PA13	SYS_JTMS-SWDIO	n/a	n/a	n/a	TMS
	PA14	SYS_JTCK-SWCLK	n/a	n/a	n/a	TCK
TIM9	PE5	TIM9_CH1	Alternate Function Push Pull	No pull-up and no pull-down	Low	
USART3	PD8	USART3_TX	Alternate Function Push Pull	No pull-up and no pull-down	Very High	STLK_RX [STM32F103CBT6_PA3]

IP	Pin	Signal	GPIO mode	GPIO pull/up pull down	Max Speed	User Label
					*	
	PD9	USART3_RX	Alternate Function Push Pull	No pull-up and no pull-down	Very High *	STLK_TX [STM32F103CBT6_PA2]
USB_OTG_FS	PA8	USB_OTG_FS_SOF	Alternate Function Push Pull	No pull-up and no pull-down	Very High *	USB_SOF [TP1]
	PA9	USB_OTG_FS_VBUS	Input mode	No pull-up and no pull-down	n/a	USB_VBUS
	PA11	USB_OTG_FS_DM	Alternate Function Push Pull	No pull-up and no pull-down	Very High *	USB_DM
	PA12	USB_OTG_FS_DP	Alternate Function Push Pull	No pull-up and no pull-down	Very High *	USB_DP
Single Mapped Signals	PA10	USB_OTG_FS_ID	Alternate Function Push Pull	No pull-up and no pull-down	Very High *	USB_ID
	PB3	SYS_JTDO-SWO	n/a	n/a	n/a	SW0
GPIO	PC13	GPIO_EXTI13	External Interrupt Mode with Rising edge trigger detection	No pull-up and no pull-down	n/a	USER_Btn [B1]
	PB0	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	LD1 [Green]
	PE10	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	USER_LD4
	PE12	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	USER_LD3
	PE14	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	USER_LD2
	PE15	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	USER_LD1
	PB14	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	LD3 [Red]
	PG6	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	USB_PowerSwitchOn [STMPS2151STR_EN]
	PG7	GPIO_Input	Input mode	No pull-up and no pull-down	n/a	USB_OverCurrent [STMPS2151STR_FAULT]
	PB7	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	LD2 [Blue]
	PE0	GPIO_Output	Output Push Pull	No pull-up and no pull-down	Low	USER_LD5

8.2. DMA configuration

nothing configured in DMA service

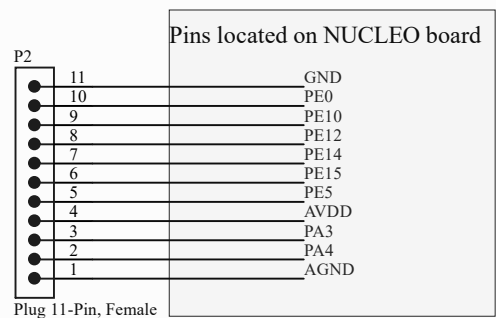
8.3. NVIC configuration

Interrupt Table	Enable	Preenmption Priority	SubPriority
Non maskable interrupt	true	0	0
Hard fault interrupt	true	0	0
Memory management fault	true	0	0
Pre-fetch fault, memory access fault	true	0	0
Undefined instruction or illegal state	true	0	0
System service call via SWI instruction	true	0	0
Debug monitor	true	0	0
Pendable request for system service	true	0	0
System tick timer	true	0	0
ADC1, ADC2 and ADC3 global interrupts	true	0	0
TIM1 break interrupt and TIM9 global interrupt	true	0	0
USART3 global interrupt	true	0	0
TIM7 global interrupt	true	0	0
PVD interrupt through EXTI line 16	unused		
Flash global interrupt	unused		
RCC global interrupt	unused		
EXTI line[15:10] interrupts	unused		
USB On The Go FS wake-up interrupt through EXTI line 18	unused		
Ethernet global interrupt	unused		
Ethernet wake-up interrupt through EXTI line 19	unused		
USB On The Go FS global interrupt	unused		
FPU global interrupt	unused		

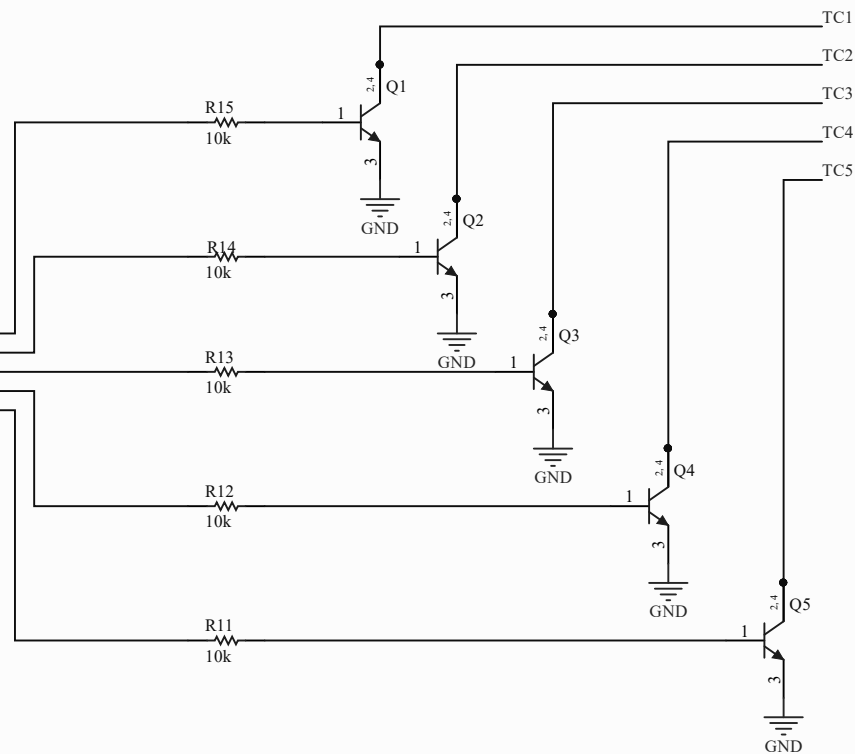
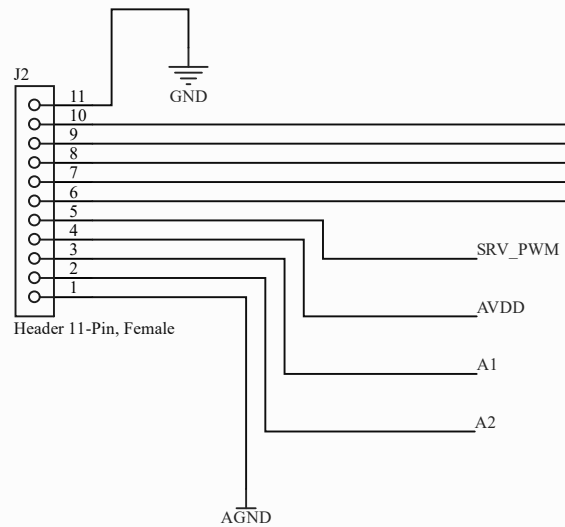
* User modified value


9. Software Pack Report

Plug with connections to Nucleo Board



Header on PCB



Title <i>Signals Connector</i>			<i>PUT Poznan Drawn by M. Sadowski Checked by A. Kolodziejczyk</i>	
Size: A4	Number:2	Revision:		
Date: 13.02.2020	Time: 12:34:45	Sheet2 of 2		
File: SignalConnector.SchDoc				