Realizujący: **Michał Kowalik (284687)**
Data: **3.01.2019r.**

Sprawozdanie

Projekt implementacji bibliotek w języku C do: obsługi modułu Bluetooth HC-05, drivera UART oraz obsługi protokołu do przesyłu danych zapisanych na karcie pamięci Data Loggera CAN (moja praca inżynierska)

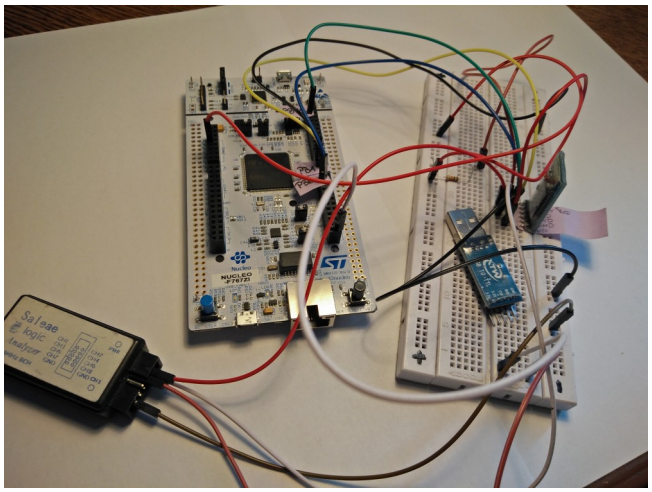
Pełny kod projektu dostępny jest pod adresem:
github.com/mkowalik/DataLogger_Bluetooth

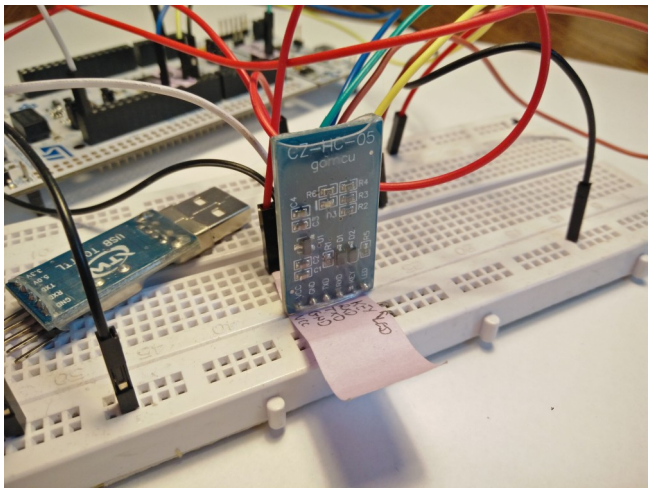
1. Cel projektu:

Celem projektu było stworzenie biblioteki do obsługi modułu Bluetooth HC-05, która do komunikacji pomiędzy mikrokontrolerem, a modułem bezprzewodowym wykorzystuje UART. Częścią projektu było również stworzenie drivera UART, który jest wykorzystywany w bibliotece do modułu Bluetooth. Kolejnym etapem było stworzenie i implementacja protokołu do przesyłu danych (przy pomocy modułu HC-05) użytych w mojej pracy inżynierskiej (logger danych do magistrali CAN). Oprogramowanie było wdrożone na mikrokontrolerze z rodziny STM32 F7 (mikrokontroler STM32F767ZIT6) znajdującego się na płytce rozwojowej (Nucleo-144 STM32F767) połączonego przy pomocy płytki stykowej z modułem HC-05 v2. Przetestowana została również maksymalna szybkość komunikacji pomiędzy mikrokontrolerem, a modułem HC-05.

2. Zestaw testowy

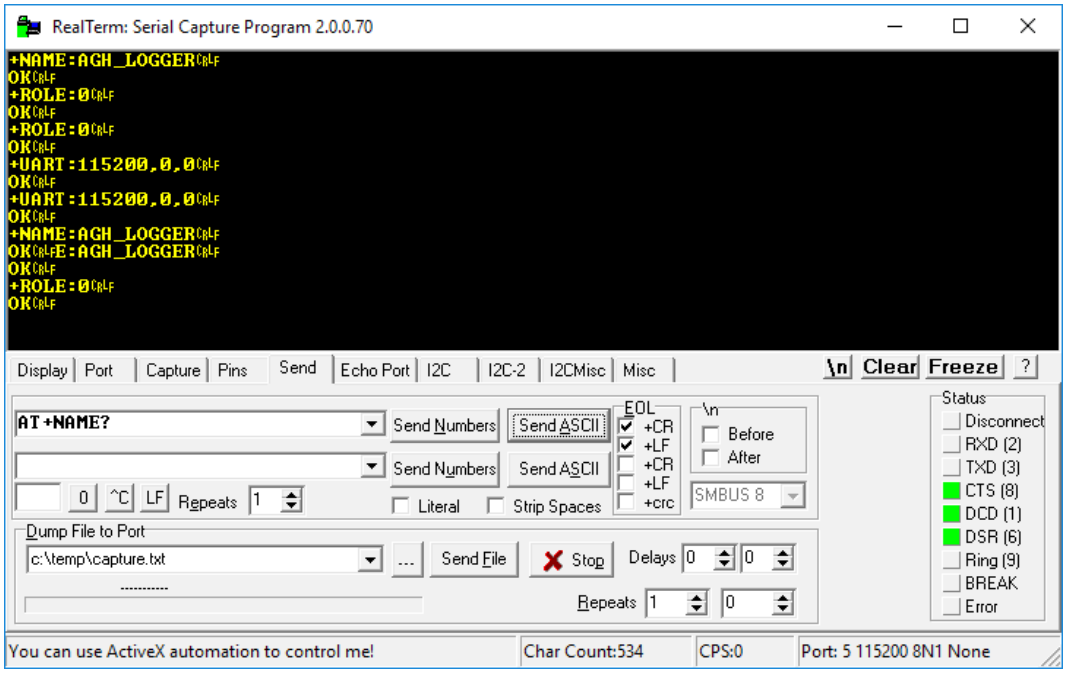
Zestaw składa się z płytki STM32F767 Nucleo-144, modułu HC-05 v2 oraz płytki stykowej. Do testowania wykorzystałem oscyloskop oraz analizator stanów logicznych. Do pierwszych testów wykorzystałem też konwerter USB-TTL tworzącą na PC wirtualny port COM. W celu testu komunikacji bluetooth użyłem aplikacji Serial Bluetooth Terminal na telefon komórkowy z systemem Android.

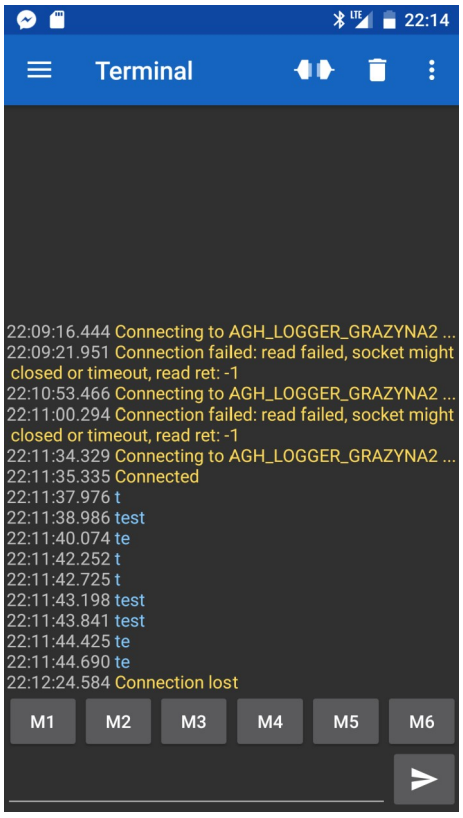




3. Pierwsze testy przy użyciu konwertera USB to TTL (UART)

Pierwsze testy protokołu i komunikacji wykonałem przy pomocy konwertera USB to TTL podłączonego to komputera PC i programu RealTerm w którym testowałem komendy AT występujące w dokumentacji, sposoby zmiany trybu pracy modułu HC-05 oraz podstawowe możliwości w zakresie przesyłu danych. Dane przesyłałem pomiędzy telefonem z system Android wykorzystując aplikację Serial Bluetooth Terminal. Poniżej screenshoty z programu RealTerm oraz aplikacji na telefon.





4. Implementacja biblioteki do obsługi moduł HC-05

Biblioteka zawiera strukturę w której zapisany jest stan drivera (pełni rolę klasy wykorzystując paradygmat programowania obiektowego), szereg funkcji (pełniących role metod tej klasy) do obsługi modułu oraz predefiniowane polecenia AT wysyłane do modułu.

Struktura posiada handler do drivera UART, handler do drivera wyjścia podłączonego do piny KEY modułu HC-05, stan w jakim się znajduje, wartość częstotliwości komunikacji UART, bufor do przesyłu i odbioru danych, oraz tablicę callbacków wraz z ich argumentami:

```
typedef struct {
    UartDriver_TypeDef*                 pUartDriver;
    DigitalOutDriver_TypeDef*           pKeyPinDriver;
    HC05Driver_State_TypeDef            state;
    uint32_t                            dataBaudRate;
    uint8_t                             buffer[HC05_BUFFER_SIZE];
    UartDriver_CallbackIterator_TypeDef callbacksIterators[HC05_DRIVER_MAX_CALLBACK_NUMBER];
    void (*callbacks[HC05_DRIVER_MAX_CALLBACK_NUMBER])(uint8_t byte, void* pArgs);
    void*                               callbackArgs[HC05_DRIVER_MAX_CALLBACK_NUMBER];
} HC05Driver_TypeDef;
```

W kodzie są predefiniowane następujące komendy AT wykorzystywane przez niżej opisane funkcje biblioteki:

```
#define HC05_AT_PREFIX_COMMAND              "AT"

#define HC05_RESTORE_ORGL_AT_COMMAND        "+ORGL"

#define HC05_GET_UART_AT_COMMNAND           "+UART?"
#define HC05_SET_UART_AT_COMMNAND           "+UART="
#define HC05_GET_UART_AT_COMMNAND_RESPONSE  "+UART:"

#define HC05_GET_PSWD_AT_COMMNAND           "+PSWD?"
#define HC05_SET_PSWD_AT_COMMNAND           "+PSWD="
#define HC05_GET_PSWD_AT_COMMNAND_RESPONSE  "+PSWD:"

#define HC05_GET_NAME_AT_COMMNAND           "+NAME?"
#define HC05_SET_NAME_AT_COMMNAND           "+NAME="
#define HC05_GET_NAME_AT_COMMNAND_RESPONSE  "+NAME:"

#define HC05_GET_ROLE_AT_COMMAND            "+ROLE?"
#define HC05_SET_ROLE_AT_COMMAND            "+ROLE="
#define HC05_GET_ROLE_AT_COMMNAND_RESPONSE  "+ROLE:"

#define HC05_AT_RESET_COMMAND               "+RESET"

#define HC05_GET_STATE_AT_COMMNAND          "+STATE?"
#define HC05_GET_STATE_AT_COMMNAND_RESPONSE "+STATE:"

#define HC05_INITIALIZED_RESPONSE           "INITIALIZED"
#define HC05_READY_RESPONSE                 "READY"
#define HC05_PAIRABLE_RESPONSE              "PAIRABLE"
#define HC05_PAIRED_RESPONSE                "PAIRED"
#define HC05_INQUIRING_RESPONSE             "INQUIRING"
#define HC05_CONNECTING_RESPONSE            "CONNECTING"
#define HC05_CONNECTED_RESPONSE             "CONNECTED"
#define HC05_DISCONNECTED_RESPONSE          "DISCONNECTED"

#define HC05_COMMAND_TERMINATION            "\r\n"

#define HC05_SET_OK_COMMAND_RESPONSE        "OK"

#define HC05_COMMAND_TRIM_SIGN              '\n'

#define HC05_START_UP_DELAY_MS              1000
#define HC05_AT_MODE_DELAY_MS               30
```

Pierwsza z funkcji służy do inicjalizacji modułu. Wprowadza ona moduł w stan AT i konfiguruje nazwę urządzenia, docelową prędkość komunikacji, hasło (PIN), rolę urządzenia, a na koniec resetuje urządzenie do pracy do trybu przesyłu danych:

```
72 HC05Driver_State_TypeDef HC05Driver_init(HC05Driver_TypeDef* pSelf, HC05Driver_Role_TypeDef role, \
73         UartDriver_TypeDef* pUartDriver, DigitalOutDriver_TypeDef* pKeyPinDriver, uint32_t baudRate,
74         char* name, uint16_t password);
```

Kolejna zawierają kod wysyłający komendę AT testującą połączenie z modułem Bluetooth oraz czekający na odpowiedź. W zależności od odpowiedzi (OK lub ERROR) zwraca odpowiednią wartość statusu.

```
76 HC05Driver_Status_TypeDef HC05Driver_sendTestATCommand(HC05Driver_TypeDef* pSelf);
```

Kolejna wysyła komendę AT przywracającą stan domyślny modułu Bluetooth.

```
77 HC05Driver_Status_TypeDef HC05Driver_sendRestoreDefualtCommand(HC05Driver_TypeDef* pSelf);
```

Następne przy pomocy komend AT pobierają z HC-05 aktualną prędkość komunikacji (baudRate) pomiędzy mikrokontrolerem a modułem, lub ustawiają nową szybkość komunikacji na jedną z dostępnych.

```
79 HC05Driver_Status_TypeDef HC05Driver_getBaudRate(HC05Driver_TypeDef* pSelf, uint32_t* pRetBaudRate);
80 HC05Driver_Status_TypeDef HC05Driver_setBaudRate(HC05Driver_TypeDef* pSelf, uint32_t baudRate);
```

Dwie dalsze funkcje przy pomocy komend AT pobierają z HC-05 aktualne hasło (PIN) dla połączenia bluetooth, lub ustawiają nową nowe hasło.

```
82 HC05Driver_Status_TypeDef HC05Driver_getPassword(HC05Driver_TypeDef* pSelf, uint32_t* pRetPassword);
83 HC05Driver_Status_TypeDef HC05Driver_setPassword(HC05Driver_TypeDef* pSelf, uint32_t password);
```

Poniższe przy pomocy komend AT pobierają z HC-05 aktualną rolę protokołu bluetooth, lub ustawiają nową wartość tego parametru.

```
88 HC05Driver_Status_TypeDef HC05Driver_getDeviceRole(HC05Driver_TypeDef* pSelf, HC05Driver_Role_TypeDef* pRetRole);
89 HC05Driver_Status_TypeDef HC05Driver_setDeviceRole(HC05Driver_TypeDef* pSelf, HC05Driver_Role_TypeDef role);
```

Kolejne funkcje odpowiadają za blokujące wysłanie wiadomości lub za synchroniczne (jedno po drugim) wysłanie i odebranie wiadomości w wersjach: do odbioru N bajtów lub do odbierania danych aż do pojawienia się wybranego znaku.

```
93 HC05Driver_Status_TypeDef HC05Driver_sendData(HC05Driver_TypeDef* pSelf, uint8_t* data, uint16_t bytes);
94 HC05Driver_Status_TypeDef HC05Driver_sendAndReceiveDataTerminationSign(HC05Driver_TypeDef* pSelf, uint8_t* pSendData,
95         uint16_t bytesToSend, uint8_t* pReceiveBuffer, uint16_t bufferSize, uint8_t terminationSign);
96 HC05Driver_Status_TypeDef HC05Driver_sendAndReceiveDataNBytes(HC05Driver_TypeDef* pSelf, uint8_t* pSendData,
97         uint16_t bytesToSend, uint8_t* pReceiveBuffer, uint16_t bytesToReceive);
```

Istnieją również funkcje do asynchronicznego odbioru wiadomości – polega to na zarejestrowaniu funkcji callback (przyjmującej bajt i void*, zwracającej void), która zostanie wywołana po przybyciu bajtu z danymi. De facto funkcja ta nie robi nic innego, jak rejestracja callbacku w driverze UARTa.

```
98
99  HC05Driver_Status_TypeDef HC05Driver_setReceiveDataCallback(HC05Driver_TypeDef* pSelf,
100         void (*foo)(uint8_t byte, void* pArgs), void* pArgs, HC05Driver_CallbackIterator_TypeDef* pRetCallbackIterator);
101 HC05Driver_Status_TypeDef HC05Driver_removeReceiveDataCallback(HC05Driver_TypeDef* pSelf,
102         UartDriver_CallbackIterator_TypeDef callbackIterator);
```

Istnieją również trzy funkcje niedostępne dla użytkownika korzystającego z biblioteki, pozwalające przełączyć urządzenie pomiędzy trybami pracy AT i przesyłu danych. W zależności od tego, czy urządzenie zostało uruchomione w trybie AT (HardAT mode) czy było wcześniej w trybie przesyłu danych, procedura ta będzie wyglądała inaczej i wynika z dokumentacji modułu. Funkcje te wykorzystywane są przez wyżej wymienione metody dostępne dla użytkownika w celu wprowadzenia modułu HC-05 we właściwy dla danej metody stan.

```
61 static HC05Driver_Status_TypeDef HC05Driver_resetNormalMode(HC05Driver_TypeDef* pSelf);
62 static HC05Driver_Status_TypeDef HC05Driver_setATMode(HC05Driver_TypeDef* pSelf);
63 static HC05Driver_Status_TypeDef HC05Driver_setDataMode(HC05Driver_TypeDef* pSelf);
```

Urządzenie może być w kilku dostępnych stanach. Istnieje funkcja która umożliwia sprawdzenie w jakim stanie znajduje się urządzenie:

```
91 HC05Driver_Status_TypeDef HC05Driver_getState(HC05Driver_TypeDef* pSelf, HC05Driver_State_TypeDef* pRetState);
```

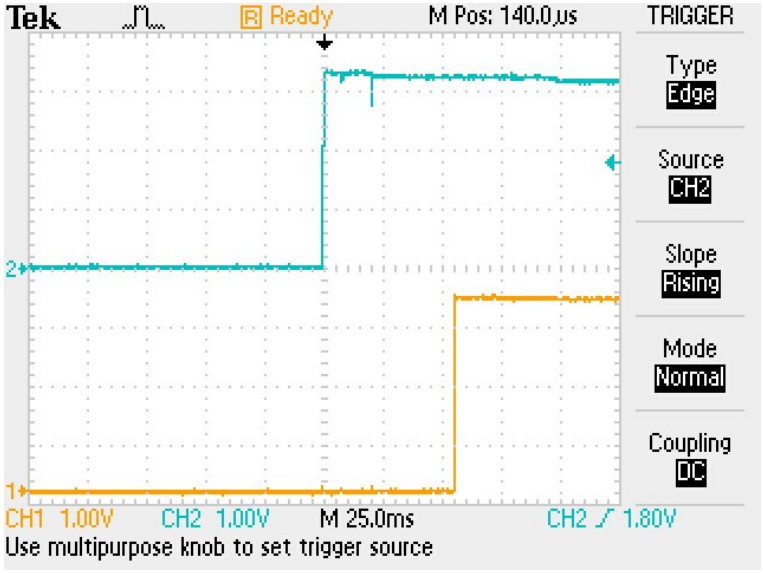
a może ono być w jednym z poniższych:

```
28
29 typedef enum {
30     HC05Driver_State_UnInitialized      = 0,
31     HC05Driver_State_HardAT,
32     HC05Driver_State_Data,
33     HC05Driver_State_Initialized,
34     HC05Driver_State_Ready,
35     HC05Driver_State_Pairable,
36     HC05Driver_State_Paired,
37     HC05Driver_State_Inquiring,
38     HC05Driver_State_Connecting,
39     HC05Driver_State_Connected,
40     HC05Driver_State_Disconnected
41 } HC05Driver_State_TypeDef;
```

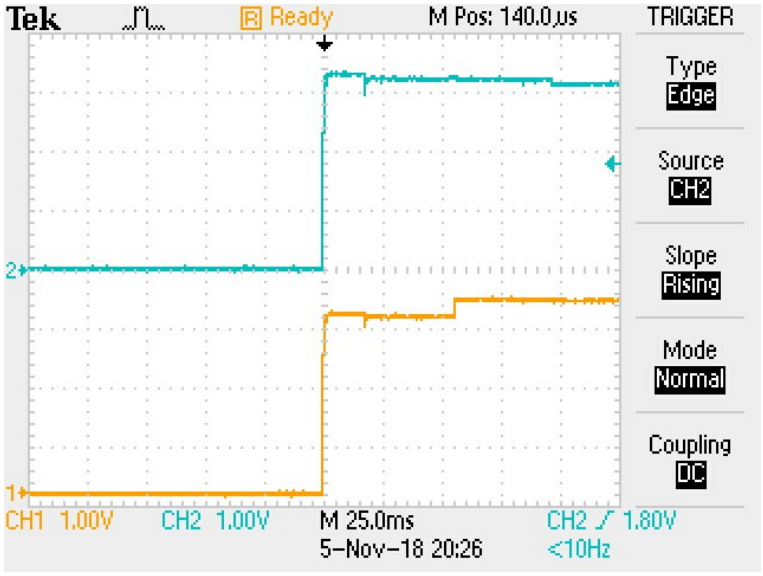
Większość stanów wynika bezpośrednio z dokumentacji (i urządzenie jest odpytywane komendą AT o to w jakim stanie się znajduje). Natomiast stan UnInitialized oznacza, że na obiekcie drivera modułu nie została wywołana funkcja _initialize. HardAT oznacza, że moduł został uruchomiony w stanie AT (poprzez podanie stanu wysokiego na nóżkę KEY modułu przed podaniem jego zasilania). Szybkość komunikacji wtedy jest ustalona na 38400bps i możliwe jest wysyłanie jedynie komend AT. Aby wyjść z tego stanu trzeba zresetować moduł bluetooth, a na nóżce KEY wtedy musi być stan niski.

5. Problem i rozwiązanie przy wprowadzaniu modułu w stan HardAT

Jako, że w zestawie testowym nie ma możliwości kontrolowania momentu uruchomienia modułu bluetooth, włączany on jest wraz podaniem zasilania do całego układu. Powodowało to problem, że przy uruchomieniu, zanim peryferia kontroloera zostały uruchomione i wysterowały stan wysoki na nóżce miktrokontrolera PB1 podłączonego do wejścia KEY w module Bluetooth, moduł ten zdążył się uruchomić w trybie normalnym.



Na powyższym zrzucie z oscyloskopu kanał 1 (pomarańczowy) podłączony jest do nóżki KEY, a kanał 2 do zasilania (niebieski). Widać tutaj, że stan wysoki na nóżce KEY pojawia się po ponad 50ms po podaniu zasilania. Problem ten został rozwiązany przez zastosowanie fizycznego rezystora pull-up podłączonego do wejścia KEY i zasilania na płytce stykowej. Wynik jest następujący:



W kolejnej wersji urządzenia konieczne jest podpięcie się pod pin enable modułu HC-05 (niewyprowadzony w tej wersji modułu) lub zastosowanie tranzystora MOS sterującego zasilaniem modułu, żeby przy resecie miktrokontrolera było możliwe w przejście w tryb hardAT. W tym momencie wejście w ten tryb możliwe jest tylko przy całkowitym odłączeniu i podłączeniu zasilania do całego układu.

6. Opis protokołu do komunikacji między loggerem a PC (koncepcja)

Celem stworzenia protokołu jest możliwość bezprzewodowego (poprzez Bluetooth) zgrania danych zapisanych na karcie microSD w DataLoggerze na komputer PC użytkownika, który te dane chce pobrać. Protokół ten ustala sposób komunikacji aplikacji na komputerze PC użytkownika z modułem bluetooth na DataLoggerze, dzięki któremu możliwe jest:

- sprawdzenie istnienia połączenia (wysłanie komunikatu ACK)
- pobranie listy plików na karcie microSD DataLoggera
- pobranie listy plików tylko z rozszerzeniem .aghlog (plików z zalogowanymi danymi)
- przesłanie wybranego pliku z loggera na komputer
- przesłanie pliku konfiguracyjnego (z roszrzeniem .aghconf)
- przesłanie z PC do loggera nowego pliku konfiguracyjnego

Protokół nie jest jeszcze w pełni zaimplementowany. Poniżej przedstawiam szkielet biblioteki realizującej opisany powyżej protokół.

```
BluetoothProtocolMiddleware_Status_TypeDef BluetoothProtocolMiddleware_init(BluetoothProtocolMiddleware_TypeDef* pSelf, HC05Driver_TypeDef* pHC05Driver);

BluetoothProtocolMiddleware_Status_TypeDef BluetoothProtocolMiddleware_sendACKResponse(BluetoothProtocolMiddleware_TypeDef* pSelf);                //0x00
BluetoothProtocolMiddleware_Status_TypeDef BluetoothProtocolMiddleware_sendListOfAllFiles(BluetoothProtocolMiddleware_TypeDef* pSelf);              //0x01
BluetoothProtocolMiddleware_Status_TypeDef BluetoothProtocolMiddleware_sendListOfLOGFiles(BluetoothProtocolMiddleware_TypeDef* pSelf);              //0x02
BluetoothProtocolMiddleware_Status_TypeDef BluetoothProtocolMiddleware_sendFile(BluetoothProtocolMiddleware_TypeDef* pSelf, char* filename);        //0x03
BluetoothProtocolMiddleware_Status_TypeDef BluetoothProtocolMiddleware_sendConfig(BluetoothProtocolMiddleware_TypeDef* pSelf);                     //0x04
BluetoothProtocolMiddleware_Status_TypeDef BluetoothProtocolMiddleware_newConfig(BluetoothProtocolMiddleware_TypeDef* pSelf, uint8_t* byteStream);  //0x05

BluetoothProtocolMiddleware_Status_TypeDef BluetoothProtocolMiddleware_worker(BluetoothProtocolMiddleware_TypeDef* pSelf);

void BluetoothProtocolMiddleware_receivedByteCallback(uint8_t byte, void* pSelf);
```

7. Opis drivera UART

Sterownik do kontrolera magistrali szeregowej również wykorzystuje paradygmat programowania obiektowego. Jest struktura odpowiadająca klasie Drivera UART, która przechowuje stan obiektu dla danego kontrolera UART, posiada zmienne takie jak wskaźnik na strukturę z biblioteki HAL odnoszącej się do konkretnego kontrolera UART w mikrokontrolerze, bufory czy tablicę callback'ów.

```
typedef enum {
    UartDriver_State_UnInitialized = 0,
    UartDriver_State_Ready,
    UartDriver_State_ChangingSettings,
    UartDriver_State_Receiving
} UartDriver_State_TypeDef;

typedef struct  {
    UartDriver_State_TypeDef    state;
    UART_HandleTypeDef*         pUartHandler;
    uint32_t                    baudRate;
    volatile uint8_t            transmitInProgress;
    volatile uint16_t           receiveBufferIterator;
    uint8_t                     receiveBuffer[UART_DRIVER_BUFFER_SIZE];
    void (*callbacks[UART_DRIVER_MAX_CALLBACK_NUMBER])(uint8_t byte, void* pArgs);
    void*                       callbackArgs[UART_DRIVER_MAX_CALLBACK_NUMBER];
    uint16_t                    readIterators[UART_DRIVER_MAX_CALLBACK_NUMBER];
} UartDriver_TypeDef;
```

Posiada on również funkcje odpowiedzialne za konfigurację, oraz wysyłanie danych (synchroniczne), wysyłanie i odbieranie danych (synchroniczne), rejestrację callbacku który ma być wywołany przy odebraniu bajtu (odbiór asynchroniczny) oraz sam callback dla odebrania lub wysłania danych z trasceivera.

```
61
62  UartDriver_Status_TypeDef UartDriver_init(UartDriver_TypeDef* pSelf, UART_HandleTypeDef* pUartHandler, uint32_t baudRate);
63
64  UartDriver_Status_TypeDef UartDriver_getBaudRate(UartDriver_TypeDef* pSelf, uint32_t* pRetBaudRate);
65  UartDriver_Status_TypeDef UartDriver_setBaudRate(UartDriver_TypeDef* pSelf, uint32_t baudRate);
66
67  UartDriver_Status_TypeDef UartDriver_sendBytes(UartDriver_TypeDef* pSelf, uint8_t* pBuffer, uint16_t bytes);
68
69  UartDriver_Status_TypeDef UartDriver_receiveBytesTerminationSign(UartDriver_TypeDef* pSelf, uint8_t* pReceiveBuffer,
70          uint16_t bufferSize, uint8_t terminationSign);
71  UartDriver_Status_TypeDef UartDriver_receiveNBytes(UartDriver_TypeDef* pSelf, uint8_t* pReceiveBuffer, uint16_t bytesToRead);
72
73  UartDriver_Status_TypeDef UartDriver_sendAndReceiveTerminationSign(UartDriver_TypeDef* pSelf, uint8_t* pSendBuffer,
74          uint16_t bytesToSend, uint8_t* pReceiveBuffer, uint16_t bufferSize, uint8_t terminationSign);
75  UartDriver_Status_TypeDef UartDriver_sendAndReceiveNBytes(UartDriver_TypeDef* pSelf, uint8_t* pSendBuffer,
76          uint16_t bytesToSend, uint8_t* pReceiveBuffer, uint16_t bytesToRead);
77
78  UartDriver_Status_TypeDef UartDriver_setReceiveDataCallback(UartDriver_TypeDef* pSelf, void (*foo)(uint8_t byte, void* pArgs),
79          void* pArgs, UartDriver_CallbackIterator_TypeDef* pRetCallbackIterator);
80  UartDriver_Status_TypeDef UartDriver_removeReceiveDataCallback(UartDriver_TypeDef* pSelf,
81          UartDriver_CallbackIterator_TypeDef callbackIterator);
82
83  UartDriver_Status_TypeDef UartDriver_receivedBytesCallback(UartDriver_TypeDef* pSelf);
84  UartDriver_Status_TypeDef UartDriver_transmitCompleteCallback(UartDriver_TypeDef* pSelf);
85
```

8. Testy maksymalnej prędkości komunikacji z modułem HC-05

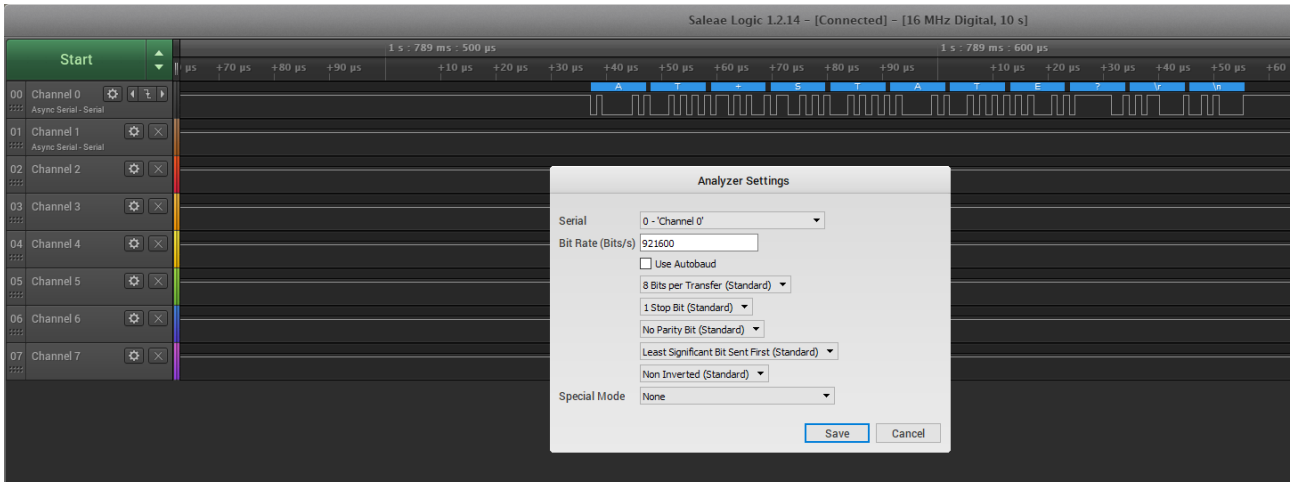
W kilku wersjach dokumentacji istnieją nieścisłości co do maksymalnej prędkości UARTa wykorzystywanego do komunikacji z modułem HC-05. Jedna określa dostępne prędkości komunikacji jako:

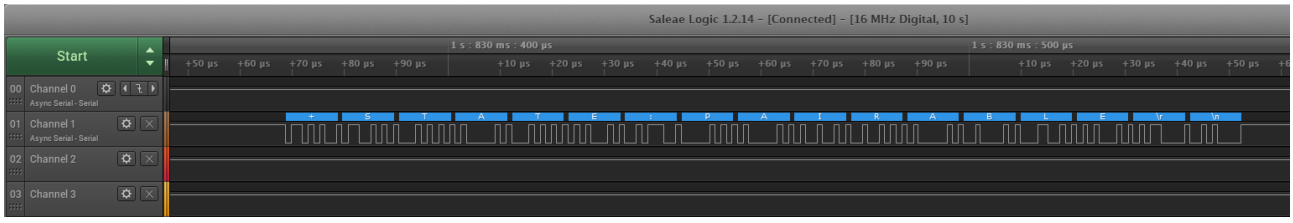
9600,19200,38400,57600,115200,230400,460800.

Druga dodaje jeszcze:

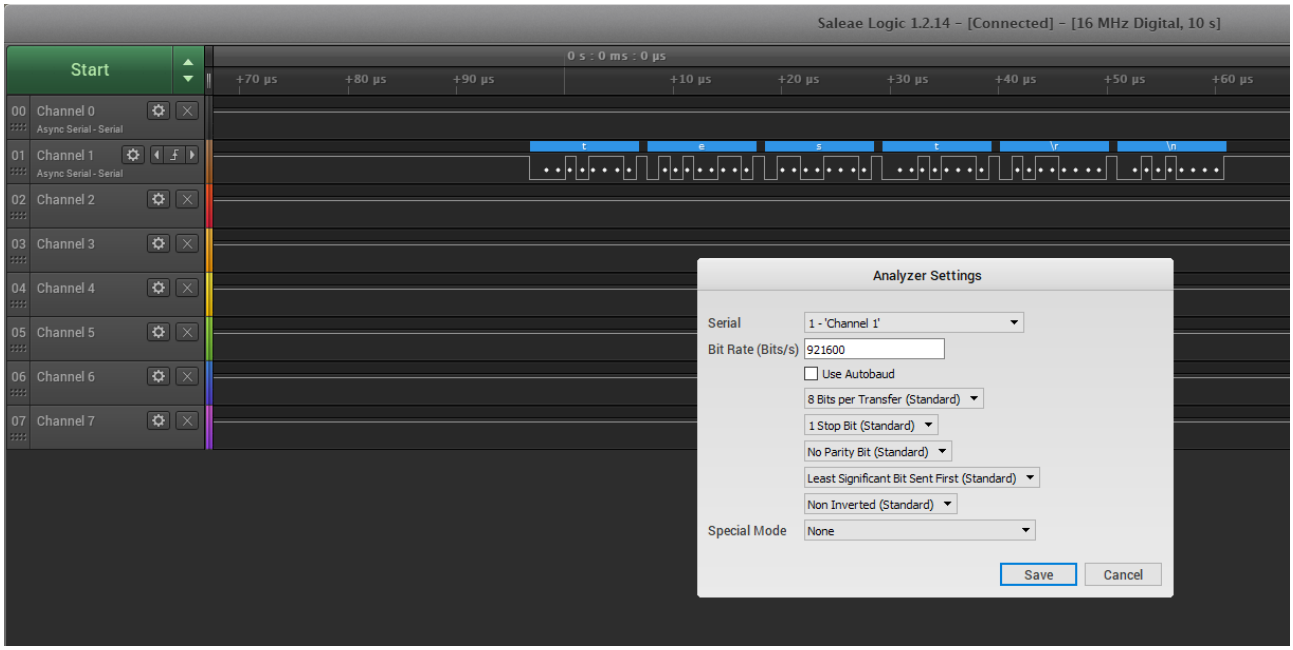
... 4800, 921600 (0,9Mbps), 1382400 (1,38Mbps).

Projekt działa w pełni poprawnie dla prędkości **921600 (0,9Mbps)** (i niższych) co ilustruje poniższy zrzut ekranu z analizatora stanów logicznych. Channel 0 jest podłączony do pinu TXD mikrokontrolera (RXD modułu bluetooth), a Channel 1 do pinu RXD mikrokontrolera (TXD modułu bluetooth):





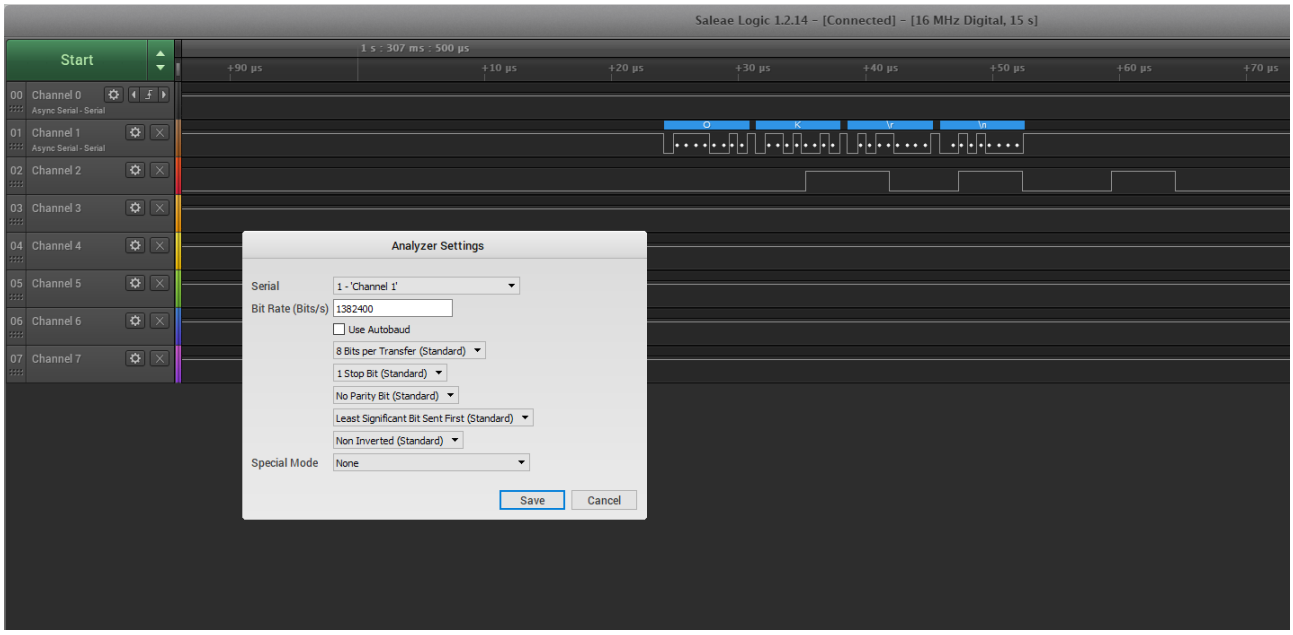
Przesył danych działa również porawnie. Poniżej wiadomość testowa wysłana z telefonu:



Prędkość **1382400 (1,38Mbps)** również poprawnie obsługiwana jest przez moduł HC-05, jednak długość procedury obsługi przerwania po otrzymaniu bajtu uniemożliwiła pracę z taką częstotliwością. Wymagałoby to przeprojektowania koncepcyjnego drivera UART. Długość obsługi przerwania sprawdziłem poprzez wystawianie stanu wysokiego na początku procedury obsługi przerwania, oraz ustawienie stanu niskiego na jej końcu. Poniżej przedstawiony jest funkcja obsługująca to przerwanie:

```
96
97  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
98      DigitalOutDriver_setHigh(&debugOut0);
99      if (huart == (&huart1)){
100         if (UartDriver_receivedBytesCallback(&uart1Driver) != UartDriver_Status_OK){
101             Error_Handler();
102         }
103     }
104     DigitalOutDriver_setLow(&debugOut0);
105 }
106
```

Poniżej zrzut ekranu dla z analizatora stanów logicznych, gdzie Channel2 podłączony do wyjścia debugOut0 – ilustruje wejście i wyjście z procedury obsługi przerwania. Zauważmy, że odpowiedź układu HC-05 jest prawidłowa (poprawna odpowiedź ‘OK\r\n’).



Zatem możliwe jest działanie układu z prędkością 1,38Mbps. Wymaga to przeprojektowania drivera UART.