Politechnika Warszawska
Wydział Elektroniki I Technik Informacyjnych
Instytut Telekomunikacji

OBIEKTY INTERNETU RZECZY

Laboratorium 1 zdalne – pomoc

1.Wprowadzenie

Prace z zagadnieniami sieciowymi z wykorzystaniem platformy Arduino UNO z nakładką Ethernet mogą być realizowane z wykorzystaniem symulatora **EBSimUnoEth**. Jest to program, który emuluje CPU zainstalowane w Arduino UNO czyli Atmega328P.
Dla potrzeb dalszej części teksu można zdefiniować pojęcie „platforma emulowana” (czyli Arduino UNO z nakładką Ethernet) i „platforma emulująca” (komputer osobisty na którym będzie uruchamiany **EBSimUnoEth**).
Proszę pamiętać, iż nie jest to idealna emulacja, istnieje wiele ograniczeń, wśród nich najważniejsze (poznane i wywnioskowane z budowy):
-szybkość wykonywania instrukcji CPU nie odpowiada żadnej istniejącej realizacji krzemowej tego procesora – innymi słowy instrukcje te wykonywane są z szybkością będącą funkcją szybkości platformy na jakiej uruchomiono emulator (np.: Windows OS, Linux, ...),
-nie wszystkie zasoby i funkcje biblioteczne Arduino IDE (opis wszystkich znajduje się pod: https://www.arduino.cc/reference/en/) są emulowane. Lista funkcji, które na platformie **EBSimUnoEth** zachowują się nietypowo:
`digitalWrite()` – widać tylko na konsoli efekt zmiany pin’u,
`delay()` – działanie nieprecyzyjne czasowo, zalecane jest stosowanie `ObirMilis()`,
`millis()` – `delay()`, zatem zalecane jest stosowanie `ObirMilis()`,
-niektórych funkcji nie zaleca się używać: `attachInterrupt()`, `detachInterrupt()`,
-niektóre zasoby peryferyjne wnętrza Atmega328P nie są emulowane: Timer 1 i 2 (Timer 0 domyślnie używane jest przez Arduino API do odmierzania czasu systemowego), przetwornik ADC, generacja PWM,
-dane wysyłane przez port szeregowy Ardunio UNO emulator wypisuje na konsoli w jakiej został uruchomiony, nie zwracając niemal uwagi na szybkość komunikacji portu szeregowego, dodatkowo warto pamiętać, że przetestowano w działaniu wyłącznie funkcje `Serial.begin()`, `Serial.print()` i `Serial.println()`.
Platforma emulacyjna EBSimUnoEth wspiera emulacje karty Ethernet w dość specyficzny sposób:
-emulacja sprowadza się do wyłącznie do komunikacji za pomocą protokółu UDP (klasa `ObirEthernetUdp`),
-emulacja zakłada podanie jako parametru wywołania numeru IP jaki ma być używane przez emulowaną platformę, co jest przydatne, gdy platforma emulująca zawiera wiele kart sieciowych o różnych numerach IP,
-komunikacja zapewniana przez `ObirEthernetUdp` jest uproszczona do obsługi jednego „strumienia” danych, innymi słowy aplikacja powinna ograniczyć się wyłącznie do wymiany datagramów z jednym zdalnym adresem IP i tylko na jednym numerze portu lokalnego – szczegóły stają się klarowne po przestudiowaniu dwóch dołączonych przykładów: `ObirUdpNtpClient_20200403.ino`, `ObirEth_udpserver_20200402.ino`),
-inne dostarczone biblioteki są wspierającymi (np.: `ObirFeatures` dostarcza dodatkowe usługi takie jak `ObirMilis()`, `ObirIPAddress` dostarcza obsługę numerów IP, gdy pozostałe: `ObirDhcp` i `ObirEthernet` dostarczono dla zgodności z całym modelem programistycznym).

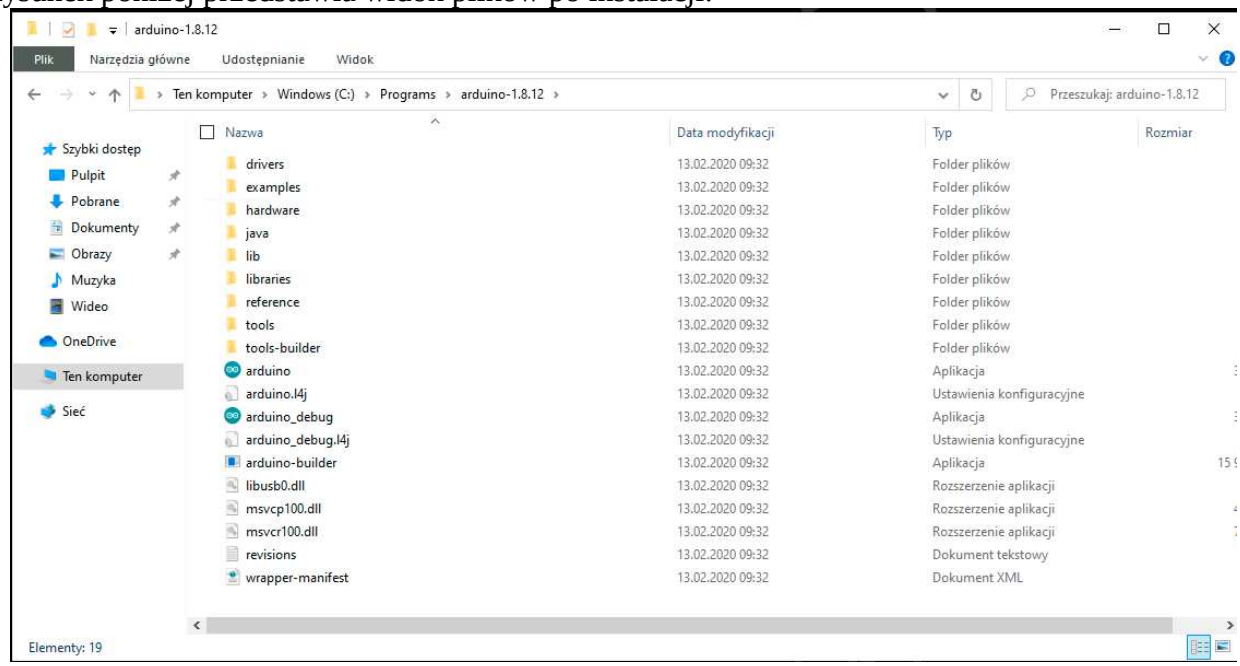
2.Instalacja środowiska

Przed rozpoczęcie prac związanych z tworzeniem kodu dla Arduino UNO emulowanego przez **EBSimUnoEth**, należy zainstalować Arduino IDE ze strony: https://www.arduino.cc/en/Main/Software (np.: dla systemu Windows: https://downloads.arduino.cc/arduino-1.8.12-windows.exe).

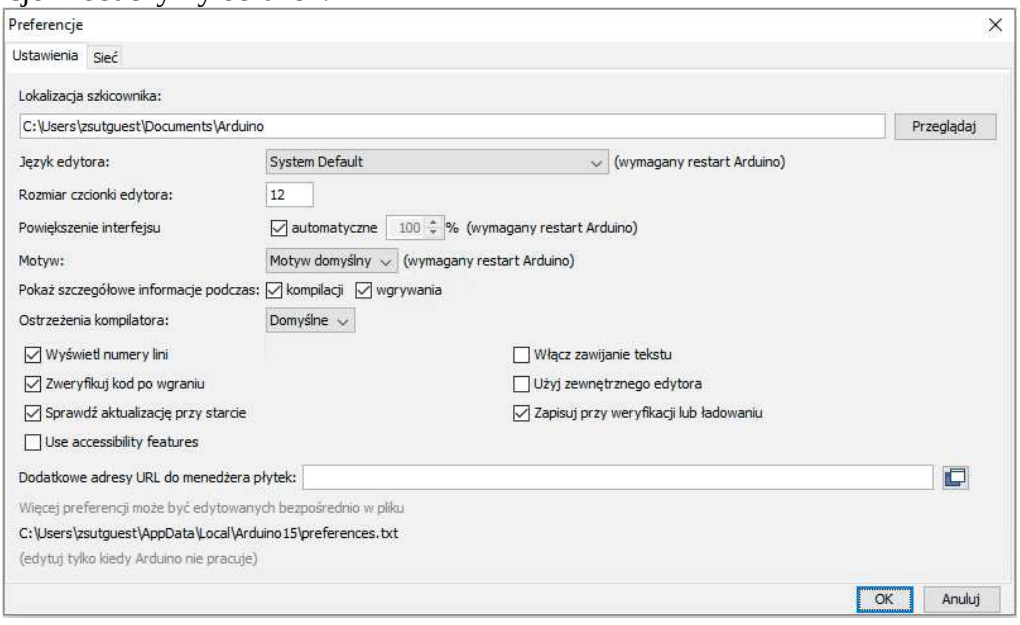
Zaleca się instalacje z instalatora w pliku EXE lub pliku ZIP i gorąco odradzamy stosowanie wersji „Windows app" ze sklepu Microsoft – problemy z ustaleniem miejsca instalacji i przechowywania elementów składowych środowiska.

Zalecanym miejscem instalacji/kopiowania środowiska Arduino IDE jest `C:\Programs` (katalog, który należy wykreować samemu). Dziwić może, że nie jest to typowe miejsce czyli np.: „`C:\Program Files`", ale wynika to głównie z dążenia do unikania spacji i znaków specjalnych w ścieżkach do wszelkich plików – takowe znaki bardzo często kompilują wiele operacji czy wręcz je uniemożliwiają.

Rysunek poniżej przedstawia widok plików po instalacji:



Po uruchomieniu Arduino IDE (plik: `C:\Programs\arduino-1.8.12\arduino.exe`) przed pierwszymi pracami należy skonfigurować i dodać biblioteki dla **EBSimUnoEth**. Na początku zdobywamy wiedzę gdzie Arduino IDE będzie składował biblioteki, wybierając „File"->"Preferencje" zobaczymy obrazek:



Na rysunku widać, że w lokalizacji `C:\Users\zsutguest\Documents\Arduino`, będą przechowywane szkice, na szczęście od pewnego wydania środowiska Arduino, również biblioteki są umieszczane w tym katalogu: `C:\Users\zsutguest\Documents\Arduino\libraries`. Do tego katalogu trzeba skopiować zawartość katalogu ObirEthernet dostarczonego poprzez system GIT (opis poniżej).

3.Instalacja systemu kontroli wersji GIT

Istnieje na rynku cała plejada narzędzi do kontroli wersji oprogramowania, dla potrzeb studentów przygotowano system GIT. Dla systemu Windows można pobrać klienta z: https://git-scm.com/download/win, obecnie wspieranym i zalecanym jest „Git-2.26.0-64-bit.exe". Proszę go zainstalować najlepiej w katalogu C:\Programs\Git, używając domyślnych ustawień podczas procesu instalacji.

4.Użytkowanie systemu GIT

Przed rozpoczęciem pracy z systemem GIT należy w dowolnym miejscu utworzyć miejsce, np.: wydając polecenie CMD (aplikacja dostępna przez tzw. „lupkę") i przechodząc do katalogu za pomocą polecenia:

```
cd C:\Users\zsutguest\
```

a następnie utworzyć katalog np.:

```
mkdir temp
```

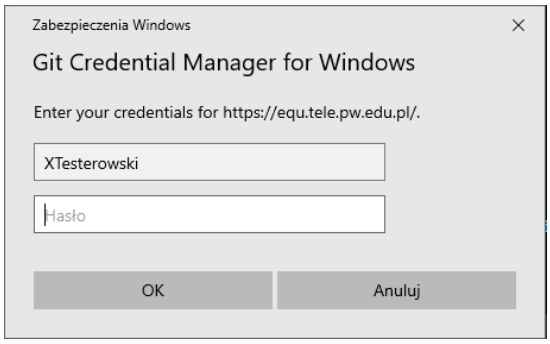
i wejść do niego:

```
cd temp
```

Mając gotowe miejsce możemy pobrać przygotowaną porcję plików klonując zdalne repozytorium (dane uwierzytelniające przesłane zostaną później, tu używany danych hipotetycznego i nie istniejącego użytkownika – tj. login: XTesterowski, haslo: Xtest2020):

```
git clone https://XTesterowski@equ.tele.pw.edu.pl/obir/XTesterowski
```

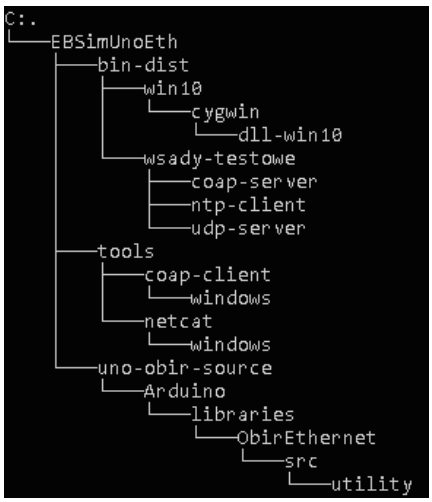
Podczas klonowania polecenie GIT zada pytanie o dane uwierzytelniające, które należy wpisać do okienka jak to:



Proszę pamiętać, że zachowanie systemu GIT może na niektórych instalacjach nieznacznie się różnić – np.: w systemie Linux pytanie o dane uwierzytelniające mogą być podawane w linii poleceń. Dodatkowo proszę pamiętać, aby nie używać metody kopiuj-wklej, z nieznanych przyczyn proces taki kończy się porażką autoryzacji.

Uwaga! polecenie „`git clone`" wykonuje się tylko raz w danej lokalizacji – czytaj zawsze w pustym miejscu, czyli tam gdzie nic jeszcze nie było wykonywane/modyfikowane.

Struktura pierwotna pobranych plików powinna być następująca:



Inne przydatne polecenia systemu GIT:
a)dodatnie wszystkich swoich poprawek do lokalnego repozytorium (domyślnie po sklonowaniu danych ze zdalnego repozytorium pracujemy na własnej kopii), korka w poleceniu oznacza – dodaj wszystkie zmodyfikowane pliki:

```
git add .
```

Po wydaniu polecenia możemy zobaczyć czy wszystkie nasze poprawione pliki zostały uwzględnione, wydając polecenie

```
git status
```

b)zatwierdzanie zbioru poprawek (niezbędne, aby system poprawnie rozpoznawał kiedy i co zostało uznane za zrobione):

```
git commit –a –m „Zdawkowy i jednoznaczny opis co zostalo wykonane”
```

Po wydaniu powyższego polecenia możemy swoje poprawki i prace (nowe pliki) „wypchnąć” do zdalnego repozytorium:

```
git push
```

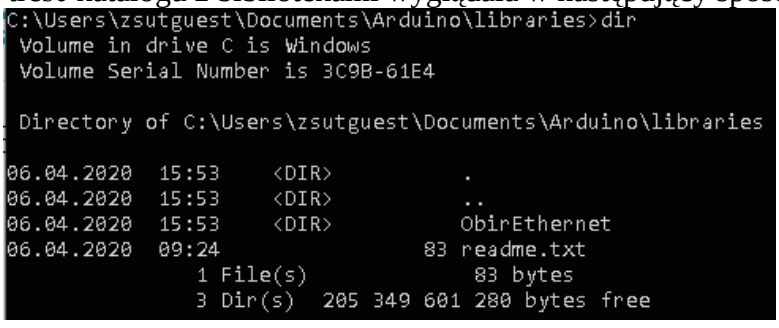
Więcej na temat systemu GIT można znaleźć w Internecie, w tym na stronie:
https://git-scm.com

5.Kopiowanie bibliotek wspierających tworzenie kodu dla platformy EBSimUnoEth

Jak wcześniej wspomniano docelowe miejsce przechowania bibliotek w środowisku Ardunio IDE to: `C:\Users\zsutguest\Documents\Arduino\libraries`

tam też trzeba skopiować katalog pobrany podczas klonowania t.j.:
`EBSimUnoEth\uno-obir-source\Arduino\libraries\ObirEthernet`

Tak, aby wynikowa treść katalogu z bibliotekami wyglądała w następujący sposób:

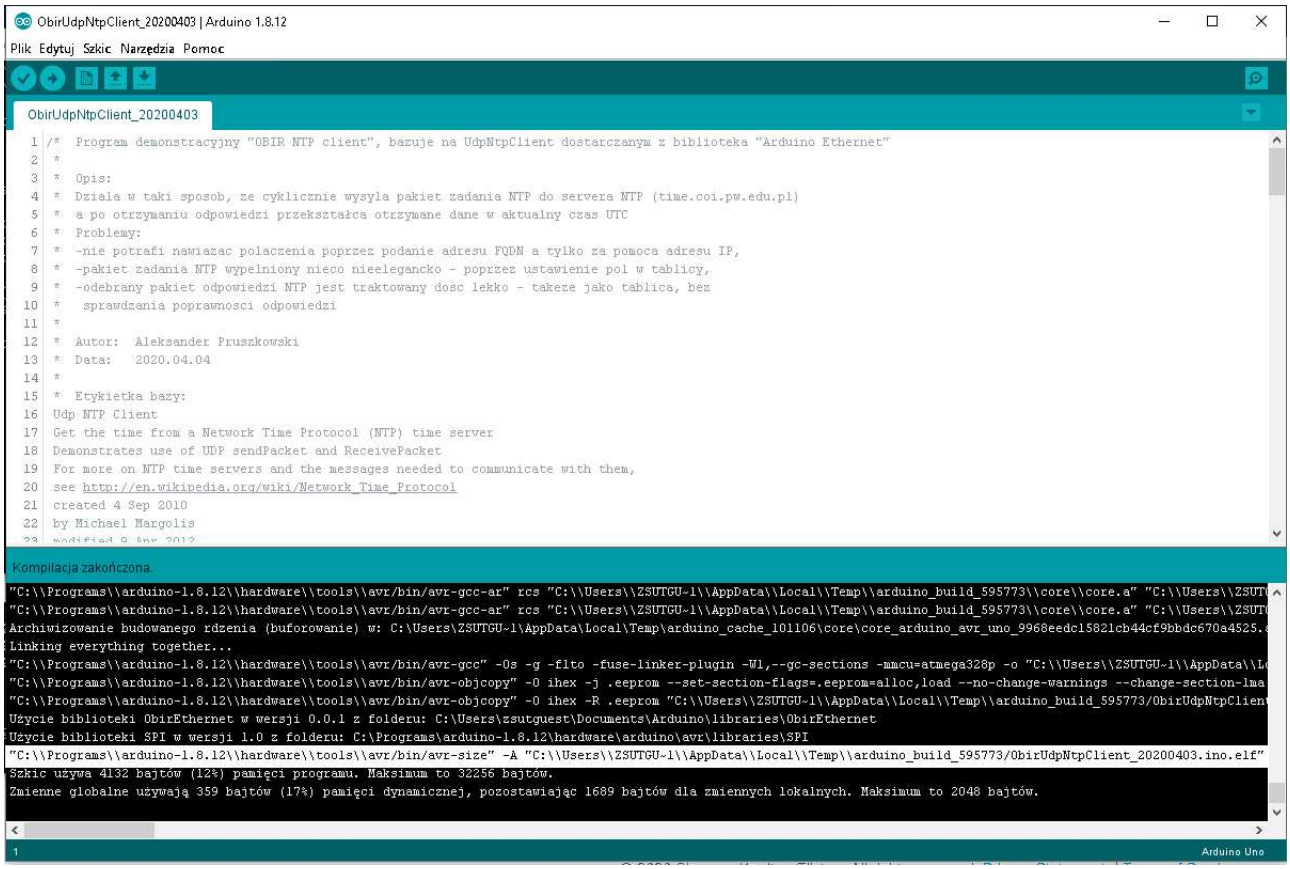


6.Tworzenie programów dla platformy EBSimUnoEth

Dla celów dydaktycznych w repozytorium GIT umieszczono także dwa pliki:
`ObirEth_udpserver_20200402.ino`
`ObirUdpNtpClient_20200403.ino`

Stanowią one pomoc w testowaniu swojego środowiska, jak i w tworzeniu nowego kodu. Zatem aby utworzyć plik z obrazem pamięci kodu (tzw. firmware) dla emulatora **EBSimUnoEth**, trzeba jeden z tych plików wkleić do okienka głównego Arduino IDE a następnie wybrać „Narzędzia” ->

„Płytka” -> „Arduino Uno” i dokonać kompilacji „Szkic” -> „Weryfikuj/Kompuiluj”. Na poziomie kompilacji platforma Arduino Uno i **EBSimUnoEth** są ze sobą zgodne.
Jednak aby można było znaleźć efekt kompilacji należy śledzić ten proces. Wybierając „Plik” -> „Preferencje” a następnie w polach koło tekstu „Pokaż szczegółowe informacje podczas:” zaznaczając „kompilacji” i „wgrywania” (po czym wyłączamy i włączamy ponownie Arduino IDE), sprawimy, że Arduino IDE w dolnej części swojego okna będzie przedstawiać szczegóły kompilacji.
Rysunek poniżej pokazuje jak Arduino IDE przedstawia w dolnym oknie (na czarnym tle) proces kompilacji szkicu. W jednej z ostatnich linii system pokazał: „`Szkic używa 4132 bajtów (12%) pamięci programu. Maksimum to 32256 bajtów`”. Jest to bardzo cenna informacja o stopniu zużycia zasobów, jednakże na tym etapie bardziej interesująca jest dla nas jedna długa linia ukazana powyżej (nad linią o zużyciu pamięci, na rysunku wyróżniona inwersją kolorów), tj.:
`"C:\\Programs\\arduino-1.8.12\\hardware\\tools\\avr/bin/avr-size" -A "C:\\Users\\ZSUTGU~1\\AppData\\Local\\Temp\\arduino_build_595773/ObirUdpNtpClient_20200403.ino.elf"`



W prawej części tej linii możemy przeczytać, że narzędzia kompilacji umieszczą pliki wynikowe w katalogu: `C:\\Users\\ZSUTGU~1\\AppData\\Local\\Temp\\arduino_build_595773`

Tam też trzeba szukać plików HEX niezbędnych dla dalszej pracy. Niestety notacja z dwoma znakami backslash („\\”) nie jest rozpoznawana poprawnie przez eksplorator systemu Windows, należy ją zamienić na tzw.: pojedyncze znaki backslash i tam szukać pliku, tj. w:
`C:\Users\ZSUTGU~1\AppData\Local\Temp\arduino_build_595773`

Po odnalezieniu pliku `ObirUdpNtpClient_20200403.ino.hex`, który zawiera obraz pamięci kodu, należy go skopiować do katalogu pobranego narzędziem GIT a zawierającym program **EBSimUnoEth**, tj. do miejsca:
`C:\Users\zsutguest\temp\XTesterowski\EBSimUnoEth\bin-dist\win10\cygwin`

Umieszczony tam plik `run_me.bat`, zawiera wszystkie niezbędne do uruchomienia emulacji dane. Prolog tego pliku zawiera niezbędną i wykonywaną na chwilę modyfikację ścieżki systemowej,

niezbędnej dla prawidłowego działania programu **EBSimUnoEth**. Jedyne na co warto zwrócić uwagę to ostatnia linia:

```
EBSimUnoEth.exe -ip 10.17.0.238 ObirUdpNtpClient_20200403.ino.hex
```

Należy ją zmodyfikować zgodnie z ustawieniami platformy emulującej, tj. numerem IP karty sieciowej. W przykładzie jest to `10.17.0.238`. Odkrycie numeru IP kary sieciowej można np.: wykonać z linii pleceń przez wydanie polecenia: „`ipconfig /all | more`”. Ważne na tym etapie jest, aby komputer posiadał przynajmniej jedną kartę sieciową podłączoną do Internetu, niestety interfejs loopback (z reguły o IP: 127.0.0.1) podobnie jak interfejsy klasy TAP/TUN czy tzw.: „VirtualBox Host-Only” nie zapewniają takiej łączności.

Po kliknięciu na `run_me.bat` zostanie otwarte okienko tekstowe a w nim uruchomiony emulator. Rozpoczęcie pracy przez emulator **EBSimUnoEth** pokazuje poniższa ramka:

```
EBSimUnoEth.exe -ip 10.17.0.238 ObirUdpNtpClient_20200403.ino.hex
Platform NIC IP: 10.17.0.238 will be used.
new_feature_init()
udp_init()
obir eth ntp client init...
[C:\Users\zsutguest\Documents\Arduino\ObirUdpNtpClient_20200403\ObirUdpNtpClient
_20200403.ino, Apr  6 2020, 15:57:14]
UDP local port: 8888, sockfd=3
NTP request sending...
Seconds since Jan 1 1900 = 3795334635
Unix time = 1586345835
The UTC time is 11:37:15
NTP request sending...
Seconds since Jan 1 1900 = 3795334695
Unix time = 1586345895
The UTC time is 11:38:15
NTP request sending...
Seconds since Jan 1 1900 = 3795334755
Unix time = 1586345955
The UTC time is 11:39:15
...
```

Zaleca się uruchamianie **EBSimUnoEth** z linii poleceń, jest to szczególnie pomocne, aby przeciwdziałać zamknięciu okna w którym działa emulator, gdy ten niespodziewanie zakończy swoją prace. Dane wygenerowane po takim nagłym zakończeniu pracy emulatora są bardzo cenne dla jego twórcy i osób z nim współdziałających – pomaga usunąć usterki za których przyczyną program **EBSimUnoEth** zakończył niespodziewanie swoją pracę.

Po uruchomieniu program **EBSimUnoEth** emuluje w nieskończoność program dla **Arduino UNO**, jednak możliwe jest przerwanie jego pracy za pomocą klasycznego „CTRL-C”.

Proszę pamiętać, że bardziej wyszukane używanie **EBSimUnoEth** jest także możliwe, np.: uruchomienie dwóch lub więcej instancji tego programu. Jest to jednak możliwe, gdy każda z tych instancji będzie uruchamiana z plikiem HEX odpowiednio dla niej spreparowanym. W czasie preparowania takich plików HEX należy zwrócić uwagę na konieczność dzielenia się zasobami – np.: tutaj stosem IP. I tak gdy więcej niż jedna instancja emulatora będzie próbowała zestawić komunikację z wykorzystaniem protokołu UDP stosując ten sam numer lokalnego portu UDP, prawdopodobnie tylko pierwsza z nich otrzyma zasób sieciowy i wykona poprawnie emulowany kod. Dla przykładu wsad: `ObirUdpNtpClient_20200403.ino.hex`, uruchomi się wyłącznie na pierwszej instancji, pozostałe będą emulowane błędnie, nie uda się im nic wysłać drogą sieciową.

Na dzień dzisiejszy emulator **EBSimUnoEth** jest poprawnie działającym produktem klasy „pre-Beta”. Tworząc własne szkice należy o tym pamiętać. Wszelkie uwagi odnośnie działania proszę kierować pod adres „apr12@o2.pl”, wstawiając na początku tematu słowo EBSimUnoEth oraz w treści oprócz opisu problemu jako załączniki dołączając kod własnego szkicu i zrzut danych wypisanych na konsoli (najlepiej w postaci tekstowej). Po otrzymaniu takich danych możliwa będzie diagnoza ewentualnej usterki emulatora oraz jego poprawienie.

7.Współpraca z protokołem UDP/IP poprzez emulowaną kartę Ethernet

Nakładka Ethernet dla Arduino UNO, zawiera układ W5100, który jest odpowiedzialny za komunikację poprzez łącze Ethernet. W przypadku **EBSimUnoEth** łączność ta jest emulowana, a odpowiednie funkcjonalności zapewniają klasy `ObirEthernet` i Obir`EthernetUdp`. Dla zbudowania prostego serwera UDP, należy wykonać następujące kroki:

a) dołączyć odpowiednie pliki nagłówkowe:

```
#include <ObirEthernet.h>
#include <ObirEthernetUdp.h>
```

b) określić adres MAC swojego urządzenia w 6 bajtowej tablicy `mac`, tutaj dla emulatora to:

```
byte mac[]={0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x01};
```

Należy tu pamiętać, że używane numery MAC powinny być unikatowe – podczas pracy z emulatorem może wydawać się to nie potrzebne to jednak dla zgodności z oryginalną platformą należy o tym zawsze pamiętać gdyż w przeciwnym razie urządzenia podpięte do tej samej sieci mogą sobie nawzajem przeszkadzać podczas komunikacji.

c) utworzyć obiekt typu `ObirEthernetUDP` niezbędny do realizacji połączeń UDP:

```
ObirEthernetUDP Udp;
```

d) w funkcji setup() uruchomić bibliotekę ObirEthernet oraz dla wygody i zgodności można także przekazać na konsolę portu szeregowego otrzymany dla Arduino UNO z serwera DHCP adres IP:

```
ObirEthernet.begin(mac);
Serial.println(ObirEthernet.localIP());
```

e) ustalić port UDP, inicjalizując zmienną `localPort`. Numer portu podaje każdej grupie prowadzący.

```
unsigned int localPort = ...;
```

f) uruchomić obsługę protokołu UDP na porcie zapisanym w zmiennej `localPort`.

```
Udp.begin(localPort);
```

g) cyklicznie np.: w ciele funkcji **`loop()`**, przetwarzać pakiety UDP odebrane przez platformę kopiując ich treść do zmiennej `packetBuffer` o długości `MAX_BUFFER`:

```
int packetSize=Udp.parsePacket();
if(packetSize){
      int r=Udp.read(packetBuffer, MAX_BUFFER);
      ...
}
```

Metoda **`Udp.parsePacket()`** pomaga w obsłudze pakietów UDP i zwraca długość właśnie odebranego pakietu. Ważne odnotowania jest, iż to wywołanie nie czeka na dane; gdy ich nie ma, zwróci wartość 0, co będzie oznaczać, że nie odebrano żadnego pakietu. Dodatkowo należy pamiętać, iż wielkość datagramu zwrócona przez **`Udp.parsePacket()`** (tu skopiowana do zmiennej: `packetSize`) może być większa od wartości `MAX_BUFFER`, w takim przypadku programista musi „skonsumować" za pomocą metody **`Udp.read()`** także resztę datagramu lub mieć świadomość, iż traci część otrzymanych danych.

h) gdy zajdzie potrzeba - odsyłać pakiet UDP z treścią przekazaną w zmiennej `sendBuffer` o długości `len`:

```
Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
int r=Udp.write(sendBuffer, len);
Udp.endPacket();
```

Proszę pamiętać, że metoda **`Udp.write()`** zwraca liczbę przyjętych przez warstwę niższą danych do wysłania (tu: zmienna ‘`r`’), w niektórych przypadkach liczba ta może być mniejsza od zmiennej ‘`len`’, wtedy programista także musi pamiętać o rozwiązaniu tego przypadku i ewentualnym przesłaniu pozostałej części danych.
Dodatkowo proszę pamiętać, że `Udp.remoteIP(), Udp.remotePort()` – mogą zwrócić wartości błędne w przypadku gdy węzeł nie otrzymał żadnego datagramu UDP. W takim przypadku należy „recznie” ustalić adres IP i numer portu zdalnego węzła do którego pragniemy wysłać datagram UDP. Dla przykładu może to wyglądać tak:

```
#define UDP_REMOTE_PORT      4321
#define PACKET_BUFFER_SIZE   32
ObirIPAddress address_ip=ObirIPAddress(10,0,4,1);
unsigned char packetBuffer[PACKET_BUFFER_SIZE]={....};
...
Udp.beginPacket(address_ip, UDP_REMOTE_PORT);
int r=Udp.write(packetBuffer, PACKET_BUFFER_SIZE);
Udp.endPacket();
```

Po wywołaniu ostatniej funkcji (endPacket) datagram UDP zostanie wysłany do maszyny o IP: 10.0.4.1 na jej numer portu 4321.

8.Literatura uzupełniająca

1.http://faculty.nptu.edu.tw/~clchen/NetworkPrograming/Bee_NetProg.html, ostatnia odsłona 2020.03.26