Aplikacje Mikrokontrolerów
Sprawozdanie z projektu -
Komunikacja z płytką przez interfejs Ethernet przy użyciu stosu TCP/IP – Współbieżny Serwer Daytime

Autorzy:
Kamil Kaliś, Radosław Skałbania, Szymon Stolarski

09.09.2020

1. Linki

a. Repozytorium SVN:
   TODO
b. Zdjęcia lub filmik dokumentujący w chmurze:
   TODO

2. Wstęp

Celem wykonanego projektu było zaimplementowanie łączności poprzez TCP/IP klienta ze zdalną płytką działającą jako serwer.
Do realizacji została użyta platform **STM32 Nucleo-F401RE**, wyposażona w rdzeń **ARM Cortex M4** 84 MHz. Ponieważ płytka sama w sobie nie posiada interfejsu Ethernet, użyta została nakładka ze złączem Ethernet, tak zwane *Arduino Shield* z modułem **Wiznet W5100**.

Kiedy serwer pozostaje uruchomiony, otwarte jest gniazdo sieciowe, które nasłuchuje na zdalnych klientów, którzy zechcą połączyć się z serwerem. Kiedy pojawia się żądanie na takie połączenie, serwer oddelegowuje obsługę połączenia do kolejnego gniazda i ciągle nasłuchuje na kolejne połączenia.
Takie rozwiązanie pozwala na podłączenie i obsługę wielu zdalnych klientów jednocześnie.

3. Użyty sprzęt i oprogramowanie

- **WIZNET W5100**

Moduł Wiznet W5100 służy jako kontroler Ethernetowy, zawiera w sobie stack TCP/IP oraz zintegrowane warstwy fizyczną PHY oraz MAC. Został wyposażony w 4 gniazda sieciowe oraz w 16 KB bufor. Bufor jest równomiernie rozłożony na łączność Rx oraz Tx, z możliwością rozdystrybuowania 8 KB na 4 gniazda. W5100 to stabilne i wydajne rozwiązanie dla łączności Ethernetowej bez systemu operacyjnego.

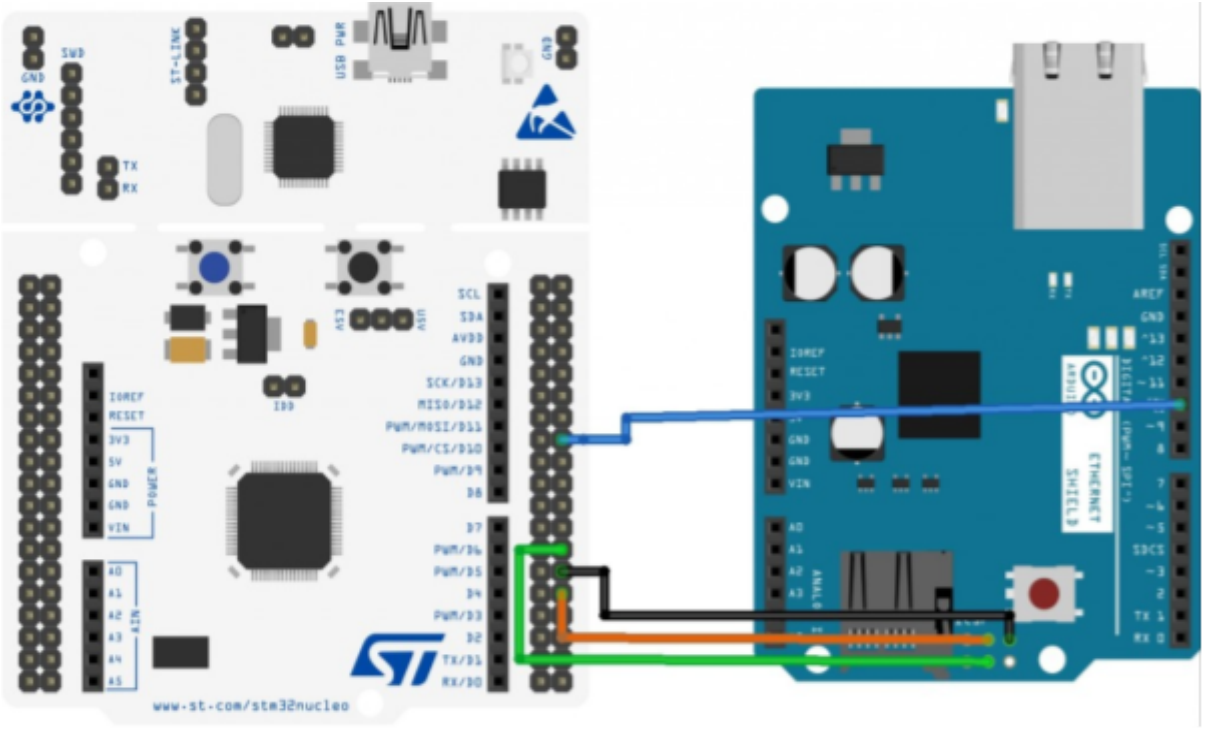
- **STM32 Nucleo-F401RE**

Wyposażona w MCU ARM Cortex M4 pozwala na budowanie wysokiej jakości i wydajności aplikacji różnego rodzaju. Aby zapewnić możliwość współbieżnej obsługi klientów, na platformie Nucleo działa system operacyjny **FreeRTOS**, który zarządza komunikują poprzez protokół komunikacyjny SPI z modułem Wiznet, jak również w wydajny sposób obsługuje klientów współbieżnie.

4. Sposób połączenia modułów

Ponieważ Nucleo nie posiada odpowiednika układu ICSP Arduino, prawidłowe połączenie i komunikacja zapewniona jest przez układ peryferyjny Nucleo SPI2:

- Arduino ICSP (MISO) → Nucleo PB_14
- Arduino ICSP (MOSI) → Nucleo PB_15
- Arduino ICSP (SCK) → Nucleo PB_13
- Arduino D10 (SS) → Nucleo PB_12



Rys. 1: Połączenie układów

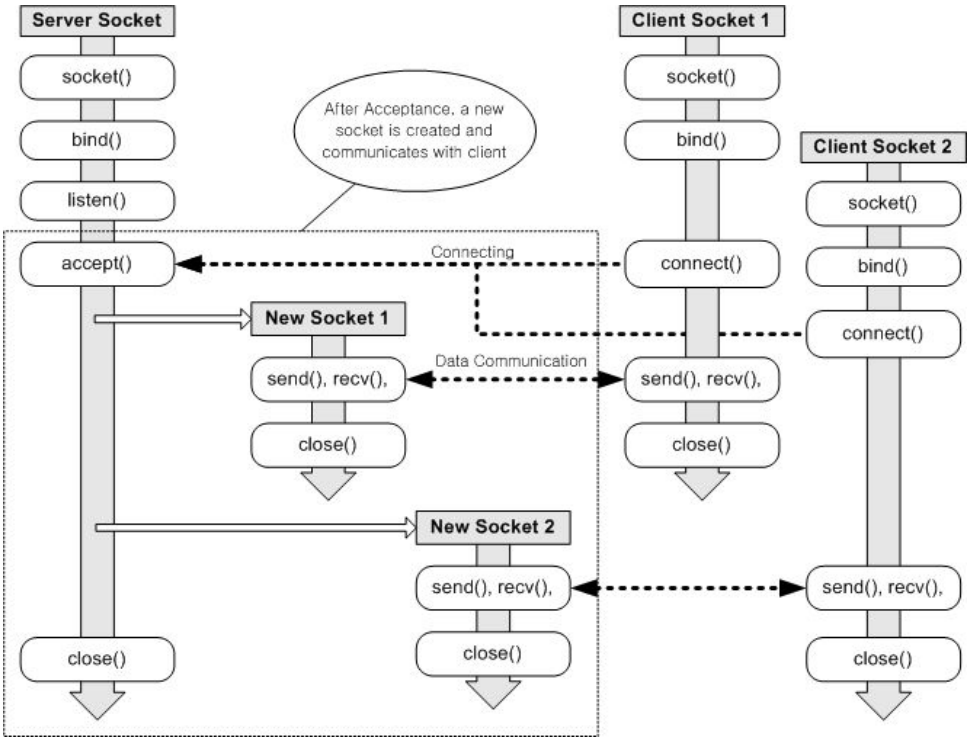
5. Sposób działania

Pierwszą rutyną wykonywaną na platformie Nucleo jest oczywiście inicjalizacja i konfiguracja wszystkich potrzebnych komponentów, takich jak:

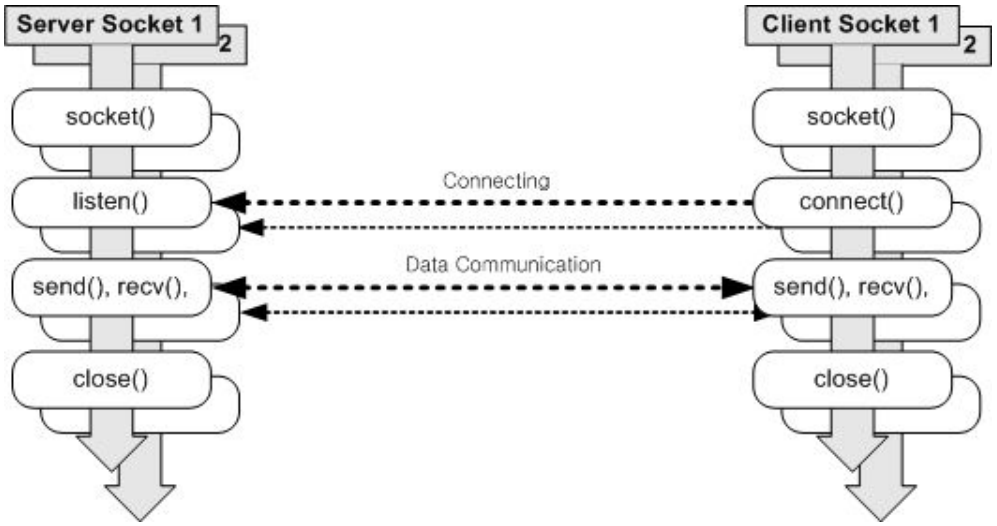
- HAL
- Zegar
- Piny GPIO
- SPI
- USART

- RTC
- I2C
- EEPROM
- Moduł Wiznet
- LED

Po udanej inicjalizacji, FreeRTOS tworzy główny wątek programu, w którym otwierane jest główne gniazdo numer 0 nasłuchujące na klientów. Kiedy otrzyma żądanie przyłączenia, z kolejki pobierane jest dostępne gniazdo (numer 1 - 3), do którego zostanie oddelegowana obsługa połączenia i świadczenia usługi. Ponieważ używane socket API udostępnione przez Wiznet działa inaczej niż standardowe Berkeley Socket API, przedstawiony projekt stara się symulować to działanie. Główny wątek odsyła do klienta numer portu utworzonego gniazda, które przejmie komunikacje, a następnie tworzy kolejne połączenie na właściwym już porcie. Poniżej przedstawione zostały diagramy obrazujące różnice między Berkeley Socket API a Wiznet Socket API:



Rys. 2: Berkeley Socket API



Rys. 3: Wiznet Socket API

Po udanym połączeniu klienta, następuje komunikacja i świadczony jest serwis *daytime* na porcie 13, gdzie aktualizowany jest czas z dokładnością do sekundy. Serwer jest w stanie na raz obsłużyć 3 klientów oraz zawiesić połączenie kolejnego klienta. Jeśli po czasie TIMEOUT=5s nie zwolni się żadne gniazdo, serwer wysyła wiadomość, która informuje drugą stronę, że powinien zerwać połączenie. Po zakończonym połączeniu, gniazdo wraca do puli (kolejki) wolnych gniazd i może zostać ponownie użyte do obsługi połączenia.

W projekcie zostały uwzględnione dodatkowe peryferia, takie jak Real Time Clock który łączy się przez i2c, dysk eeprom do zapisywania logów z działania serwera.

6. Moduły projektu

- main – główny program sterujący: Kamil Kaliś, Radosław Skałbania, Szymon Stolarski
- chip_init – rutyny inicjalizujące moduły: Kamil Kaliś
- eeprom – obsługa dysku zewnętrznego przytrzymującego logi z serwera: Szymon Stolarski
- i2c – sterownik interfejsu komunikacyjnego i2c do połączenia zewnętrznego modułu RTC: Szymon Stolarski
- led – obsługa sterowania LED: Radosław Skałbania
- retarget – przekierowanie *stdout* funkcji *printf* na USART: Szymon Stolarski
- rtc – obsługa Real Time Clock potrzebne do realizacji usługi *daytime*: Szymon Stolarski
- server – definicje głównych funkcji serwera: Kamil Kaliś, Szymon Stolarski, Radosław Skałbania

- server_utils – procedury pomocnicze dla głównego modułu serwera: Kamil Kaliś, Szymon Stolarski
- socket_queue – utworzenie kolejki i inicjalizacja dostępnymi gniazdami: Szymon Stolarski
- konfiguracja FreeRTOS: Szymon Stolarski, Radosław Skałbania
- ioLibrary_Driver – Wiznet Socket API: moduł zewnętrzny
    - socket – API do tworzenia i obsługi gniazd sieciowych
    - wizchip_conf – konfiguracja modułu Wiznet
- FreeRTOS – system operacyjny: moduł zewnętrzny
- datetime_client – dedykowany program kliencki do komunikacji z serwerem: Radosław Skałbania

Literatura

- *W5100 Datasheet – Version 1.1.6*
- *The FreeRTOS Reference Manual – V9.0.0*
- *Dokumentacja Wiznet ioLiblary*
- *Forum StackOverflow*