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

11.09.2020

1. Linki

a. Repozytorium SVN:
https://wsn.elektro.agh.edu.pl/svn/am_src/296840/projekt/

b. Zdjęcia i filmik dokumentujący w chmurze:
https://drive.google.com/drive/folders/1_ybrq468xSgzkBsY134pim4gBA--9hjm?usp=sharing

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-F446RE**

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.

- **EEPROM 21711**

  Zewnętrzna pamięć o rozmiarze 128B. Komunikacja odbywa się za pośrednictwem magistrali I2C. Dane do pamięci można zapisywać wysyłając adres komórki pamięci i dane, których może być maksymalnie 8 bajtów (rozmiar sektora).

- **miniRTC DS1307**

  Zewnętrzny RTC komunikuje się z mikrokontrolerem przez magistralę I2C. Czas jest przechowywany w rejestrach modułu w BCD.

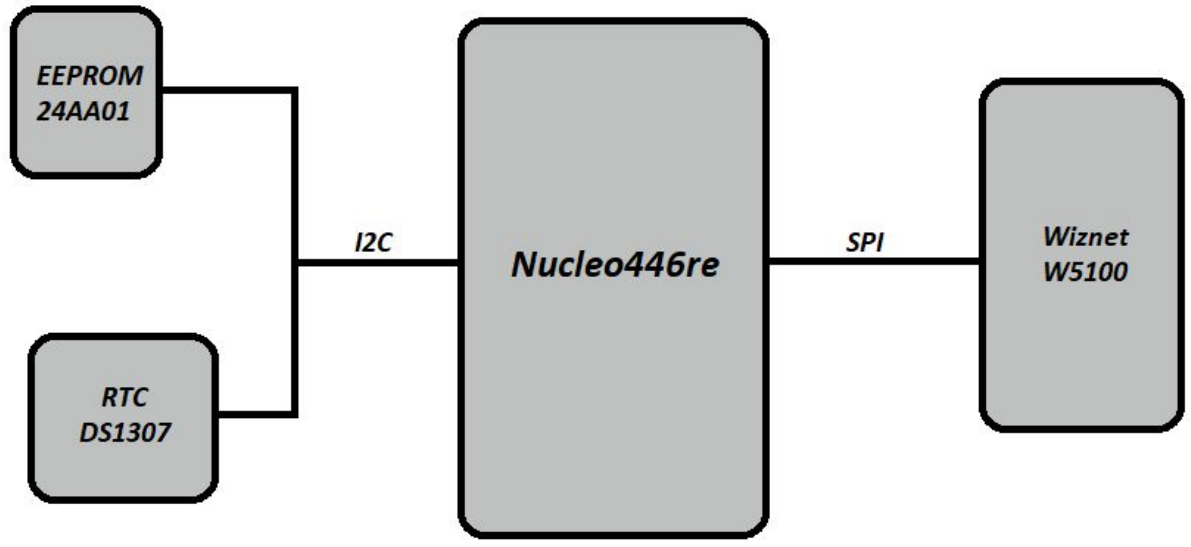
4. Sposób połączenia modułów

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

RTC i EEPROM przez I2C1

- SDA PB9
- SCL PB6



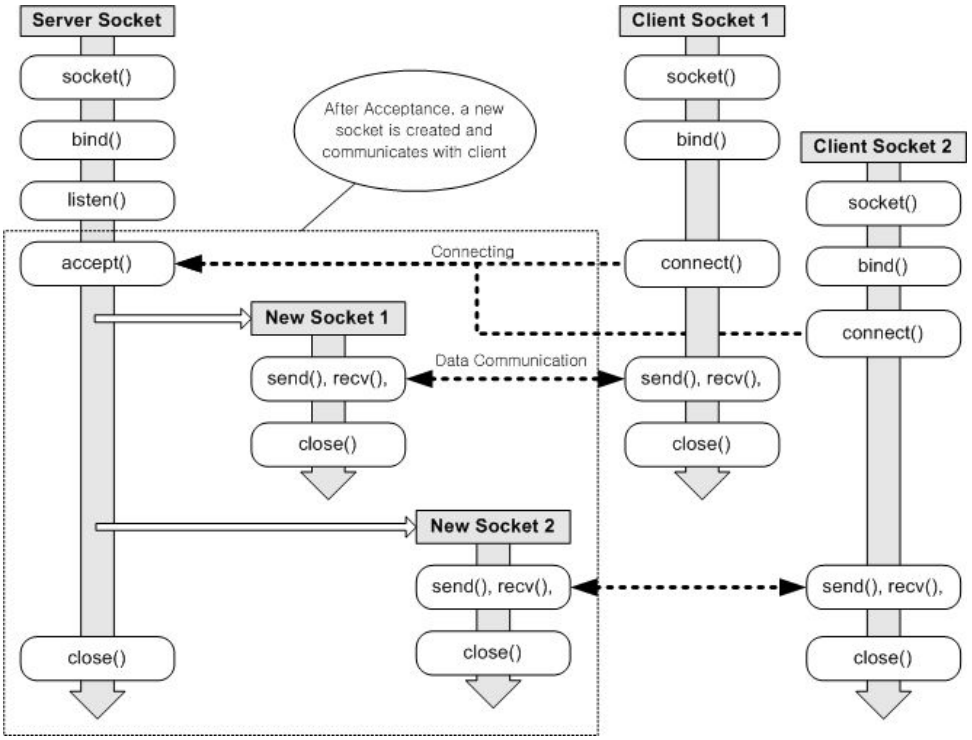
Rys. 1 Schemat blokowy

5. Sposób działania

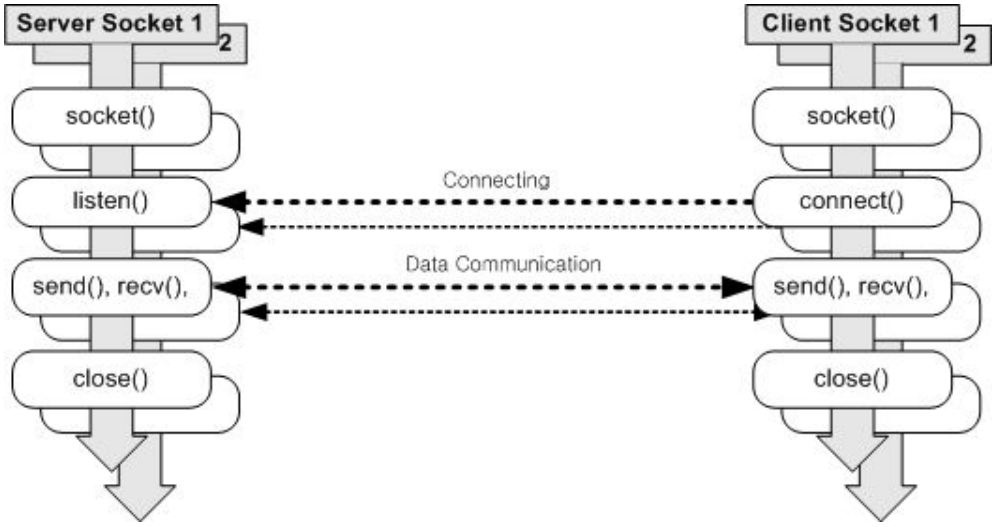
Pierwszą rutyną wykonywaną na platformie Nucleo jest oczywiście inicjalizacja i konfiguracja wszystkich potrzebnych komponentów, takich jak:

- 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.
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.
Poniżej przedstawione zostały diagramy obrazujące różnice między Berkeley SocketAPI a Wiznet Socket API:

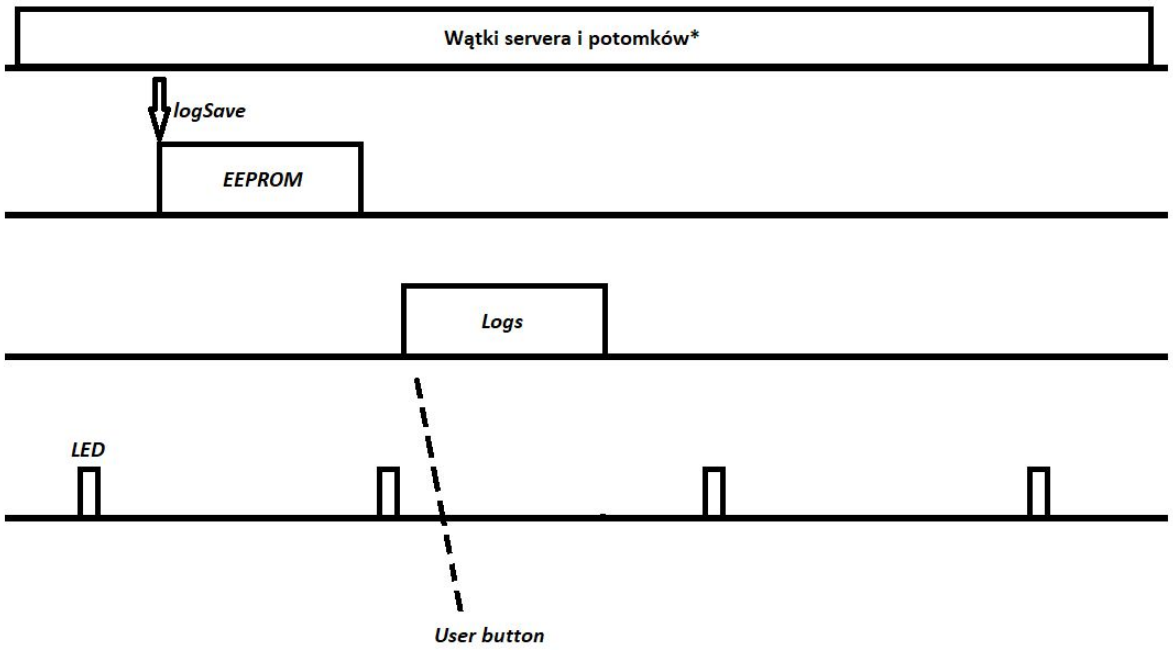


Rys. 2: Berkeley Socket API



Rys. 3: Wiznet Socket API

W projekcie zostały uwzględnione dodatkowe peryferia, takie jak Real Time Clock który łączy się przez i2c, dysk eeprom do zapisywania logów z działania serwera. W wątku EEPROM po zgłoszeniu loga przez serwer następuje zapis.
Po naciśnięciu przycisku User Button dane z pamięci EEPROM są wysyłane przez port szeregowy (wątek Logs).



Rys. 4 poglądowy wykres aktywności wątków
*szczegółowo przedstawione na rys. 2

6. Klient TCP/IP

Daytime client działa przy pomocy klasycznych sieciowych funkcji linuxowych. Do otwarcia gniazda oraz połączenia z serwerem zostały użyte funkcje *socket()* oraz *connect()*. Przy pierwszym połączeniu klient oczekuje na dane z serwera w postaci nowego portu. Odbiera je za pomocą funkcji *read()*. Następnie zamyka bieżące gniazdo i łączy się z serwerem na nowo otrzymanym porcie. Klient po ponownym połączeniu zostaje w pętli i odbiera dane w postaci aktualnej godziny dopóki proces nie zostanie przerwany lub serwer nie zakończy połączenia.

Aby poprawnie uruchomić program kliencki należy wejść do folderu *projekt/custom_client* a następnie skompilować program za pomocą komendy *make*. W terminalu uruchamiamy go za pomocą komendy *./client [-h] lub ./client [serverIP] [port] [-d (debug optional)]*.

7. I2C

I2C działa w trybie pollingu. Transmisja przebiega zgodnie ze standardem i wymogami użtych modułów. Częstotliwość SCL wynosi 400 kHz (fast mode), jest to maksymalna prędkość, którą obsługują moduły EEPROM i RTC.
Po symbolu startu wysyłany jest adres chipu i bit R/W. W przypadku odczytu sprawdzana jest flaga RXNE (Receive buffer not empty) i po każdym odebranym bajcie wysyłane jest ACK, tylko ostatni bajt nie jest potwierdzony (zgodnie z wymogami urządzeń).
W przypadku zapisu sprawdzana jest flaga TXE(Transmit buffer empty) i po każdym wysłanym bajcie sprawdzane jest jest ACK.
Zostały zaimplementowane funkcje do wysyłania; odbierania; wysyłania i odbierania (symbol restart pomiędzy wysyłaniem i odbieraniem).

8. RTC

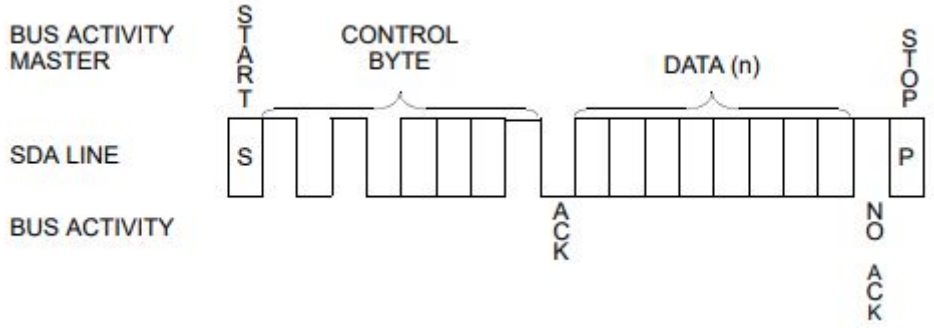
Aby zmienić stan rejestrów wewnętrznych (ustawić godzinę) trzeba wysłać dane składające się od adresu rejestru i jego zawartości.
Odczyt to wysłanie adresu rejestru i odebranie jego zawartości.

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12	10 Hour	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
		24	PM/ AM							
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

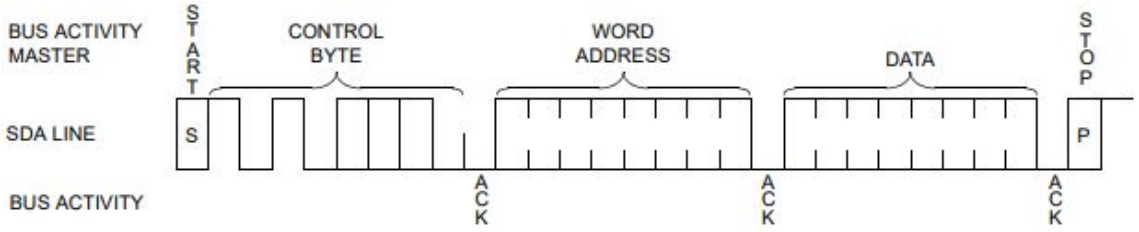
Rys. 5 Struktura pamięci RTC (zdjęcie z dokumentacji modułu RTC)

9. Pamięć EEPROM

Aby odczytać dane najpierw należy wysłać adres komórki, po tym symbol restart i można odczytać do 8 bajtów.
Przed próbą zapisu trzeba wykonać *Acknowledge polling*. Polega to na wysyłaniu na magistrali I2C adresu pamięci EEPROM z bitem R/W ustawionym na 0. Gdy chip nie ustawi bitu ACK oznacza to, że jest zajęty i nie można dokonywać zapisu i odczytu.



rys. 6 odczyt pamięci (zdjęcie pochodzi z dokumentacji EEPROM)



Rys. 7 zapis do pamięci (zdjęcie pochodzi z dokumentacji EEPROM)

10. System logów

Logi mogą być wysyłane przez port szeregowy podczas wykonywania programu przy użyciu funkcji printf, która jest przekierowana na UART.

Drugą możliwością jest zapisywanie logów w pamięci zewnętrznej EEPROM. Struktura loga składa się z numeru modułu, który go zgłosił, rodzaju zdarzenia i czasie wystąpienia (godzina i minuta).
Numer modułu to 6-cio bitowa liczba, aby go dodać należy skorzystać z funkcji *logsAddNewModule.* Wtedy nowo zdeklarowany moduł jest dopisywany do listy jednokierunkowej zawierającej wyżej opisaną strukturę.
Informacja o rodzaju zdarzenia zapisywana jest w kodzie 1 z 8. Pozwala to na przechowywanie w jednym logu wielu informacji. Dzięki temu dużo lepiej wykorzystuje się dostępne miejsce w pamięci.

Zdarzenie zgłasza się *logsAddLog* podając numer modułu i loga. Wtedy kod loga jest wpisywany w strukturze w liście jednokierunkowej.

Aby zapisać należy wywołać funkcję logsSave. Wtedy zapisywana jest godzina i minuta z RTC. Funkcja dodaje adres modułu z zapisanymi zdarzeniami do kolejki. W wątku pamięci eeprom logi są ściągane z kolejki i zapisywane w pamięci.

<table>
<tr><td colspan="2">sync</td><td colspan="6">numer modułu</td><td rowspan="2">Godzina w BCD (8 bitów)</td><td rowspan="2">Minuta w BCD (8 bitów)</td><td rowspan="2">Zgłoszone zdarzenia w kodzie 1 z 8</td></tr>
<tr><td>0</td><td>0</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td></tr>
</table>

Rys. 8 struktura bloku zapisywanego w pamięci

Blok składa się z 4 bajtów. Rozpoczyna się od dwóch bitów synchronizacyjnych. Na ich podstawie ocenia się poprawność zapisanych danych. Po nich jest 6-cio bitowy adres modułu, da bajty zawierające godzinę i minutę zapisaną w BCD i jeden bajt zawierający zgłoszone zdarzenia.

Zapisane w pamięci logi wypisywane są wciśnięciu przycisku na płytce Nucleo.

11. 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
- logs - zarządzanie zapisem i odczytem logów z/do pamięci EEPROM: Szymon Stolarski

- linked_list - tworzenie i obsługa lista jednokierunkowej: 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
- pwr, spi, uart - drivery z zasobów producenta

Literatura

- *W5100 Datasheet – Version 1.1.6*
- *The FreeRTOS Reference Manual – V9.0.0*
- *Dokumentacja Wiznet ioLiblary*
- *Dokumentacja RTC DS1307*
- *Dokumentacja EEPROM 24AA01*
- *Reference manual STM32F446*
- *Dokumentacja Nucleo446*
- *Forum StackOverflow*