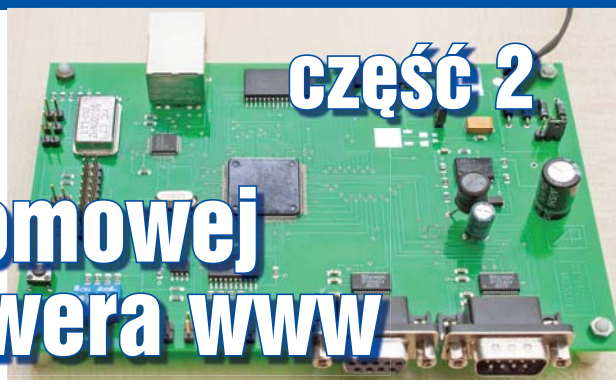
Infinity – system automatyki domowej
Oprogramowanie serwera www

kit 3232 AVT

część 2



W poprzednich odcinkach omówiliśmy budowę modułu serwera www, a ostatnio jego konfigurację. W niniejszym odcinku chciałbym przedstawić programowe szczegóły realizacji serwera. Zasadniczo materiał przeznaczony jest dla bardziej zaawansowanych, jednak do lektury zachęcam wszystkich. Jak wspomniałem wcześniej, obsługa „karty sieciowej" mikrokontrolera na najbardziej podstawowym poziomie nie jest skomplikowana. Chcę to pokazać w bieżącym numerze. Prostota i elegancja wysłania lub odebrania ramki ethernetowej jest jednak okupiona zaawansowanym rozwiązaniem sprzętowym, Niech te informacje zainspirują, zachęcą Czytelników i Czytelniczki do własnych przemyśleń i eksperymentów, gdyż to jest najlepszy sposób zdobywania wiedzy.

Tylko dla dociekliwych

Podkreślam: ta część opisu systemu automatyki domowej przeznaczona jest dla Czytelników, którzy chcą i są w stanie zrozumieć szczegóły programowej realizacji serwera. Dla mniej zaawansowanych lektura może być powodem zniechęcenia i frustracji (początkujący powinni wrócić do lektury po zakończeniu cyklu, gdy zdobędą szerszą wiedzę, gdyż nie jest wykluczone, że na łamach EdW zaprezentuję kilka bardziej szczegółowych rozważań o tematyce sieciowej). A na razie wystarczy, że zaprogramują procesor gotowym wsadem.

A teraz trudniejsze informacje dla dociekliwych. W Elportalu dostępne są wszystkie wspomniane programy i pliki, w tym program źródłowy, realizujący serwer www. Zachęcam do indywidualnej analizy, ale teraz chcę zwrócić uwagę na pewne istotne i interesujące szczegóły. Ponieważ programy są bardzo obszerne (pracowałem nad nimi wiele miesięcy), nie jest możliwe przedstawienie w EdW nawet kluczowych fragmentów, dlatego wspomniane dalej w artykule listingi 1...15 zostały umieszczone w Elportalu w postaci dodatkowych 15 małych plików tekstowych. Oto omówienie ciekawszych szczegółów:

W oprogramowaniu wprowadziłem własne nazwy podstawowych typów (**listing 1**). Do najczęściej używanych należą UCHAR (liczba całkowita 8-bitowa bez znaku – bajt), USHORT (liczba całkowita 16-bitowa bez znaku) oraz ULONG (liczba całkowita 32-bitowa bez znaku).

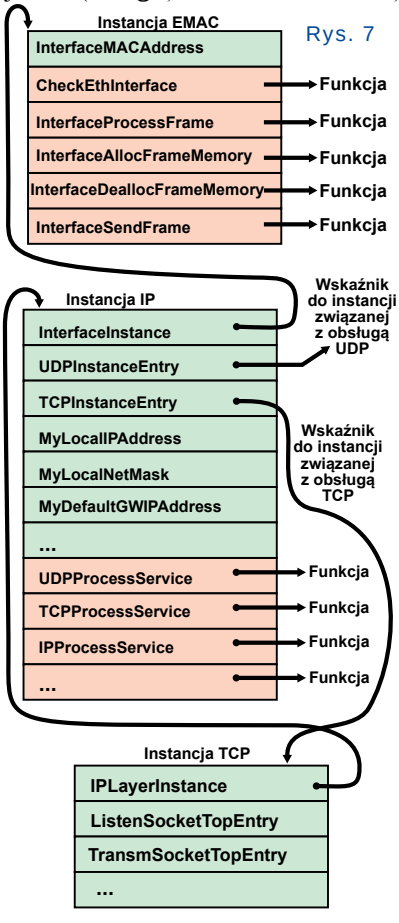
Parametry „fabryczne" są pokazane na **listingu 2**. Dane są już zapisane w formacie binarnym używanym w oprogramowaniu. Przykładowo adres IP o wartości C0A8002C w zapisie szesnastkowym po konwersji każdego bajtu do postaci dziesiętnej odpowiada adresowi IP=192.168.0.44. Podobnie zapisane są pozostałe dane o charakterze adresowym. Adres MAC zawsze jest w postaci binarnej zapisanej na 48 bitach, czyli sześciu bajtach, stąd w programie jest stosowana jako tablica sześciobajtowa.

Uruchomienie programu serwera sprowadza się do użycia funkcji *main* (**listing 3**). Nie jest tu możliwe opisanie działania wszystkich funkcji oprogramowania, ale dociekliwi Czytelnicy mogą samodzielnie przeanalizować program, gdyż komplet z programem źródłowym jest dostępny w Elportalu jako materiały dodatkowe (plik o nazwie vircom_soft.zip).

W pierwszej kolejności inicjowany jest system przerwań (wywołanie funkcji *SysInit*). Kolejna funkcja (*InitSysTime*) inicjuje działanie kolejkowania zdarzeń. Jest to istotna funkcjonalność programu odpowiedzialna za realizację pewnych czynności, które należy wykonać po określonym interwale czasu od danego momentu. Kolejne dwie funkcje (*HardwareInit* i *SoftwareInit*) konfigurują środowisko mikrokontrolera do wymaganej postaci: inicjują obsługę zewnętrznej pamięci RAM, konfigurują porty do oczekiwanej funkcji oraz inicjują zmienne programu do niezbędnej wartości początkowej. Po zainicjowaniu obu układów UART do obsługi transmisji szeregowej następuje wczytanie danych z pamięci EEPROM (funkcja *GetConfigData*). Strukturę bloku konfiguracyjnego (zapisanego w pamięci EEPROM) i realizację wspomnianej funkcji pokazuje **listing 4**, gdzie następuje wczytanie całego bloku. Program jest w stanie rozpoznać niezaprogramowaną pamięć i zainicjować pierwszy zapis do pamięci EEPROM jako domyślne parametry zastępcze („fabryczne").

W kolejnym kroku wczytany jest stan przełączników DIPSWITCH (funkcja *DIPSwitchSerwice*) i ewentualna realizacja działania wynikająca z ustawień przełączników. Obsługę pokazuje **listing 5**, gdzie po wczytaniu stanu przełączników program warunkowo przechodzi do ponownego zapisania do pamięci EEPROM parametrów fabrycznych lub przechodzi do wczytania nowej wartości adresu MAC, ewentualnie do wczytania danych o charakterze adresu IP. Po każdej operacji wykonany jest zapis nowych danych do pamięci nieulotnej oraz program jest zatrzymywany z odpowiednim komunikatem. W sytuacji, gdy nie jest rozpoznany żaden ze „stanów wyjątkowych" przełączników, program wychodzi z powyższej funkcji i realizuje kolejne działania zawarte w funkcji *main* (listing 3). Po

Rys. 7



wysłaniu w kanał szeregowy w postaci tekstowej aktualnych danych konfigurujących sieć (wywołanie funkcji *HelloMessage*), program przechodzi do zainicjowania zespołu EMAC i całej obsługi sieci (wywołane funkcji *StartNet*, która jest pokazana na **listingu 6**).

Zanim przejdę do wgłębiania się w program, muszę wyjaśnić pewien rodzaj struktury, którą określam jako instancję. W języku programowania C odpowiada to strukturze (*typedef struct*), gdzie część pól struktury zawiera specyficzne dla danej struktury dane oraz część pól jest wskaźnikami do funkcji o określonej funkcjonalności. Idea tej struktury odpowiada koncepcji obiektów w języku C++, lecz jest znacząco bardziej uproszczona. Postać instancji związanej z warstwą fizyczną pokazuje **listing 7**. Znajdują się tam klasyczne dane (jak adres MAC serwera) oraz wskaźniki do określonych funkcji związanych z odbieraniem ramek (*CheckEthInterface*), nadawaniem ramek (*InterfaceSendFrame*), przydziałem pamięci na odebrane ramki (*InterfaceAllocFrameMemory*, *InterfaceDeallocFrameMemory*) oraz przetwarzaniem odebranych ramek (*InterfaceProcessFrame*).

Kompletna obsługa sieci wymaga utworzenia dodatkowych instancji związanych z przetwarzaniem protokołów z wyższych poziomów modelu OSI. Program serwera www wymaga instancji pokazanych na **rysunku 7**, gdzie instancja EMAC (listing 7) odpowiada za fizyczną transmisję danych. Nadrzędną dla niej jest instancja związana z obsługą protokołu IP (jej strukturę można znaleźć w programach źródłowych). Zawiera ona dowiązanie do instancji EMAC oraz dowiązania do protokołów wyższych rzędów (UDP, TCP). Obok wymienionych powiązań instancja przechowuje między innymi własne parametry sieciowe (adres IP, maskę podsieci, adres IP bramy domyślnej). Wskaźniki do funkcji związanych z przetwarzaniem protokołu IP, podobnie jak w instancji EMAC, prowadzą do właściwych funkcji w programie. Przykładowo *UDPProcessService* jest odpowiedzialne za przetwarzanie ramek związanych z tym protokołem. Podobnie *TCPProcessService* prowadzi do funkcji obsługującej ramki TCP.

Powyższe rozwiązanie uelastycznia stosowanie części oprogramowania obsługującego sieć Ethernet jako niezależnej części oprogramowania mikrokontrolera ARM użytego w budowie określonego urządzenia. Dobrym tego przykładem jest właśnie serwer www, w którym nie jest używany protokół UDP (nie było takiej potrzeby). Gdyby przy rozpakowaniu odebranej ramki zostało stwierdzone, że dotyczy ona protokołu UDP, to w funkcji rozpakowującej musiałoby wystąpić wywołanie funkcji do obróbki UDP. „Sztywne" wywołanie funkcji spowoduje dodanie do kodu programu czegoś, co nie jest używane. Zamiast dodawać do programu komplet kodu, rozwiązanie z zastosowaniem wskaźników do funkcji pozwala dolinkować do programu jedynie te części, które są używane. W przypadku odebrania przez serwer pakietu UDP, gdyż nie można zabronić jakiemuś programowi mającemu dostęp do sieci lokalnej wysłania takiego pakietu do serwera, zostanie on rzecz jasna rozpoznany. Ponieważ funkcja *UDPProcessService* w tym programie ma wskaźnik pusty, taki pakiet zostanie zignorowany.

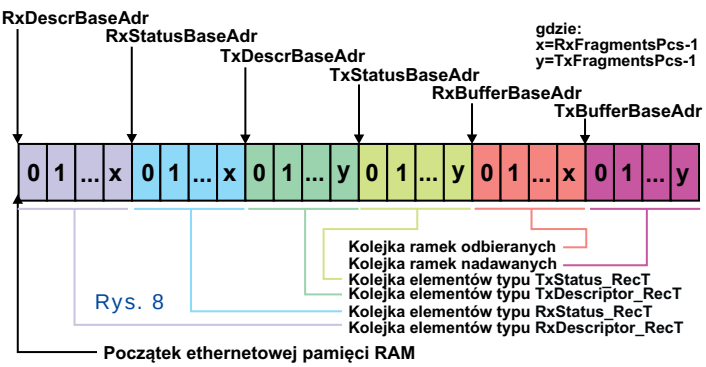
Inicjowanie i utworzenie powiązań jest zrealizowane w funkcji *StartNet* (listing 6). Instancja związana z poziomem IP jest utworzona w wyniku wywołania *InitIPLayerInstance* (tworzy dowiązanie do instancji EMAC i wypełnia wskaźniki do określonych funkcji). Niektóre z nich będą miały wskaźnik pusty jak przykładowo powiązanie z obsługą protokołu UDP. Kolejne wywołanie (w *StartNet*) funkcji *StandardAddTCPInstance* dołącza do istniejącej już instancji IP funkcje do obsługi wszystkiego co jest związane z protokołem TCP (jak na rysunku 7). Jednak cofnijmy się w analizie działania funkcji *StartNet* do początku.

Funkcja *InitPHYLayerInstance* jest odpowiedzialna na zainicjowanie obsługi sieci w warstwie fizycznej, co polega na właściwym skonfigurowaniu układu PHY oraz zespołu EMAC. Postać tej funkcji pokazana jest na **listingu 8**. Po skonfigurowaniu warstwy fizycznej sieci następuje konfiguracja „logiczna", wynikająca z adresacji IP (wywołanie funkcji *InitIPLayerInstance*), gdzie kluczową informacją jest własny adres IP oraz maska podsieci pozwalająca identyfikować „kontrahentów" należących do sieci lokalnej. Ostatnią czynnością jest zainicjowanie protokołu TCP jako wywołanie funkcji *StandardAddTCPInstance*.

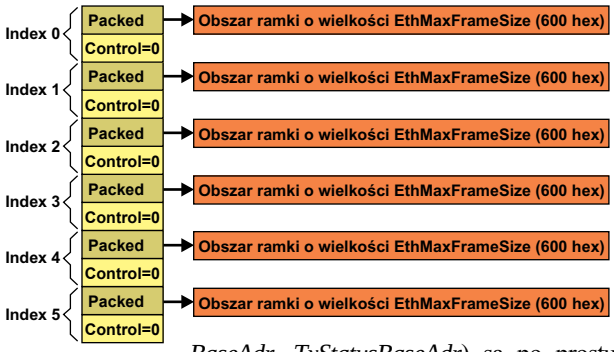
Warto przyjrzeć się bliżej procedurze inicjowania zespołu EMAC wraz z układem PHY. Zapoznanie się z obsługą zespołu sieciowego w mikrokontrolerze ARM pozwala przekonać się, że obsługa tego interfejsu na poziomie fizycznym nie jest skomplikowana, a zlecone operacje są wykonywane autonomicznie. Jedną z pierwszych czynności, jakie należy wykonać, jest włączenie zasilania dla EMAC (po sygnale reset dla mikrokontrolera domyślnie nie jest on włączony). Polega to na ustawieniu odpowiedniego bitu w rejestrze PCONP. W kolejnym kroku należy skonfigurować wyprowadzenia portu P1 do współpracy z układem PHY, czyli określić, że wybrane linie tego portu stanowią interfejs RMII do układu PHY. Polega to na odpowiednich wpisach do rejestru PINSEL2 i PINSEL3. Uważny Czytelnik zauważy, że wpis do rejestru PINSEL2 jest warunkowy. Ten mikrokontroler występuje w wersji „starej" oraz „nowej", z tego powodu realizowany jest odmienny zapis do tego rejestru. Wariant można rozpoznać po odczycie zawartości rejestru MAC_MODULEID. Po skonfigurowaniu kilku rejestrów w EMAC następuje programowe zresetowanie układu PHY jako użycie funkcji *WriteToPHY*. Przeglądając jej instrukcje, łatwo zauważyć, że zapis do PHY sprowadza się do wypełnienia określonych rejestrów EMAC (podać co i gdzie ma być zapisane) i poczekania na właściwy stan flagi potwierdzający zakończenie operacji. Podobnie przebiega odczyt z układu PHY, który można prześledzić na przykładzie odczytu identyfikatora układu PHY (wywołanie funkcji *ReadFromPHY*), czyli zapytać

układ PHY, czy „ma na imię DP83848". Analogicznie do zapisu, w operacji odczytu poprzez odpowiednie wpisy do rejestrów EMAC można zlecić wymaganą akcję i poczekać na potwierdzenie jej zakończenia. Uzyskane od układu PHY dane są do odczytania w określonym rejestrze EMAC. Prawda, że proste? W kolejnym kroku należy poinformować układ PHY o oczekiwanej prędkości komunikacyjnej (10MB/100MB). W przypadku narzuconych prędkości transmisyjnych sprowadza się to do odpowiednich zapisów do układu PHY. W przypadku autonegocjacji (jako wariantu najbardziej uniwersalnego) problem jest trochę bardziej złożony. Wstępnie układ PHY „zostaje poproszony do swoistego dogadania się" z istniejącą siecią. Z punktu widzenia elektronicznego PHY musi „złapać link" od sieci, by wywnioskować, jaka jest prędkość komunikacyjna. Ta operacja może potrwać dłuższą chwilę (z punktu widzenia programu całkiem sporą). Odczytując w pętli odpowiedni status układu PHY, wychwycony jest moment, w którym PHY zakończył autonegocjację. Pętla, w której jest odczytywany status, ma limit iteracji, gdyż w przypadku startu serwera bez połączenia z siecią (wystarczy wyjąć wtyk RJ45) program się zapętli. W dalszej kolejności program oczekuje na stan LINK (sygnalizowany również włączeniem odpowiedniej diody sterowanej bezpośrednio przez układ PHY), gdyż odczytując odpowiedni status, można uzyskać zwrotne potwierdzenie o rzeczywistej prędkości transmisyjnej. Podobna sytuacja, odczyt w pętli z limitem iteracji, przeciwdziała zapętleniu się programu. Informacja o uzyskanej prędkości jest istotna, gdyż w dalszej części wpływa na konfigurację zespołu EMAC. W sytuacji, gdy program startuje bez przyłączenia do sieci, coś trzeba założyć (że jest to sieć 100MB). Z powyższego łatwo wywnioskować, że start programu serwera www z podłączeniem do sieci przebiega trochę łagodniej. Po wpisaniu do EMAC adresu MAC (sześciobajtowego identyfikatora sieciowego na poziomie fizycznym), doprogramowaniu kilku rejestrów, zespół EMAC jest gotowy do działania. W całym ciągu instrukcji konfigurujących EMAC istotnym elementem jest wywołanie funkcji *InitEMACDescr*. Jest to bardzo istotny fragment programu, a z drugiej strony pokazuje ciekawe rozwiązanie dotyczące nadawania i odbierania ramek ethernetowych. Postać funkcji pokazuje **listing 9**.

Do obsługi sieci mikrokontroler LPC2378 ma dedykowaną pamięć RAM (poza podstawową pamięcią operacyjną) o wielkości 16kB. W obszarze tej pamięci należy programowo zainicjować odpowiednie struktury danych, które funkcjonalnie odpowiadają realizacji kolejek FIFO (**rysunek 8**). Tak, zespół EMAC mikrokontrolera ARM autonomiczne przetwarza kolejki FIFO, gdzie elementami kolejek są różne struktury danych, między innymi ramki ethernetowe o wielkości 600 hex oktetów (dla dociekliwych na temat idei kolejek FIFO przygotowałem dodatkowy dokument – *bufory cykliczne.pdf*, który można pobrać jako materiały dodatkowe z Elportalu).



Rys. 8

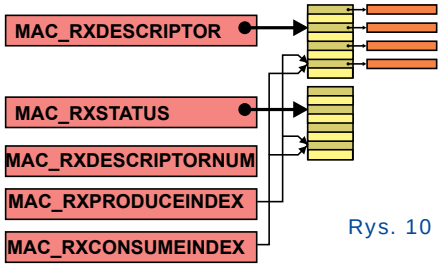


Rys. 9

Zacznę od tego, że kolejki FIFO związane z EMAC są podzielone na dwie kategorie: dotyczące nadawania danych oraz odbierania danych. Liczba elementów w buforach cyklicznych związanych z kolejkami jest określona przez dwie stałe, które można modyfikować według własnego uznania: *RxFragmentsPcs* i *TxFragmentsPcs* (listing 9). Definiują one liczbę elementów w buforach cyklicznych w odpowiednich kolejkach (tablice przechowujące elementy są indeksowane od 0 do odpowiednio *RxFragmentsPcs-1* lub *TxFragmentsPcs-1*) jak statusy czy deskryptory, których struktury definiują typy *RxDescriptor_RecT*, *RxStatus_RecT*, *TxDescriptor_RecT*, *TxStatus_RecT* (listing 9, na podstawie dokumentacji producenta UM10211 dostępnej na stronach NXP). Znając liczbę elementów w każdej kolejce oraz wielkość każdego elementu wyrażoną w bajtach, istnieje możliwość obliczenia wielkości poszczególnych obszarów i ich położenia w pamięci ethernetowej RAM.

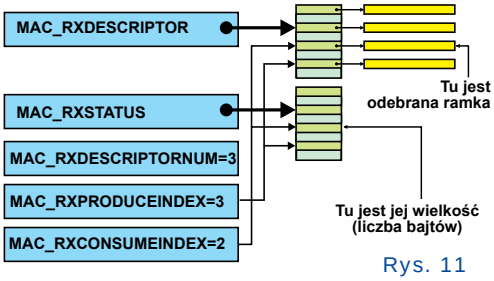
Pierwszy obszar (*RxDescrBaseAdr*, rysunek 8, **listing 10**) jest na początku obszaru pamięci RAM. Drugi (*RxStatusBaseAdr*) znajduje się o *RxFragmentsPcs*sizeof(RxDescriptor_RecT)* bajtów dalej, czyli ma adres poprzedniego obszaru (*RxDescrBaseAdr*) powiększony o wyżej określoną wielkość. Analogicznie są określone początki wszystkich pozostałych jako adres poprzedniego elementu powiększony o jego wielkość. Obszary te, w zależności od rodzaju kolejki, wymagają odpowiedniego zainicjowania zawartości. Elementy o charakterze statusowym (*RxStatusBaseAdr*, *TxStatusBaseAdr*) są po prostu wyzerowane. Pozostałe, określające bufory komunikacyjne, wymagają dowiązania do obszarów ramek (pola o nazwie *Packed*, listing 9), muszą zawierać wskazanie (adres) na obszar w pamięci, w którym jest przechowywana ramka ethernetowa. Wyjaśnia to **rysunek 9** będący kolejką FIFO ramek nadajnika (kolejki odbiornika są analogiczne). W tym przypadku jest utworzona tablica 6 elementów (taką wartość ma stała *TxFragmentsPcs*, indeksowana od 0 do 5), gdzie każde pole *Packed* wskazuje na indywidualny obszar przeznaczony na ramkę. W identyczny sposób zorganizowana jest kolejka deskryptorów odbiornika.

Tak utworzone cztery kolejki deskryptorów i statusów dla nadajnika i odbiornika są „doczepione" do rejestrów zespołu EMAC. **Rysunek 10** pokazuje powiązanie odpowiednich rejestrów (związanych z odbieraniem ramek) z właściwymi kolejkami w pamięci RAM (dowiązane są dwie kolejki, kolejne dwie dotyczące nadajnika są rozwiązane w identyczny sposób). Rejestr *MAC_RXDESCRIPTOR* wskazuje (zawiera adres) kolejkę deskryptorów określających obszary przewidziane na odbiór ramek



Rys. 10

ethernetowych (każdy element deskryptora w polu *Packed* wskazuje na indywidualny obszar przeznaczony na odebraną ramkę). Kolejny rejestr *MAC_RXSTATUS* wskazuje na kolejkę statusów związanych z operacją odbierania danych (ramek). W rejestrze *MAC_RXDESCRIPTORNUM* przechowywana jest wielkość kolejki i należy wpisać tam liczbę elementów kolejki pomniejszoną o jeden, rejestr ten dotyczy zarówno kolejki deskryptorów, jak i statusów, gdyż liczba elementów w obu kolejkach jest identyczna. Rejestr *MAC_RXPRODUCEINDEX* określa pierwsze wolne miejsce w kolejkach (tam zostanie umieszczona odebrana ramka). Podobne znaczenie ma rejestr *MAC_RXCONSUMEINDEX* wskazujący na deskryptor zawierający ramkę do przetworzenia przez program. Oba te rejestry zawierają indeks do odpowiednich tablic stowarzyszonych z kolejkami. W przypadku odbierania ramek, zespół EMAC jest „producentem" informacji zapisywanych w kolejkach (treści odebranych ramek, statusy), natomiast program serwera www jest „konsumentem" danych. W ogólnym przypadku może się tak zdarzyć, że zespół EMAC „wyprodukuje" kilka ramek, których oprogramowanie serwera nie „wygarnie". Będą one przechowywane w kolejkach (oczywiście, jeżeli zostanie przekroczona liczba nieodczytanych ramek, czyli przekroczony próg określony przez stałą *RxFragmentsPcs*, to nastąpi utrata danych). Jeżeli zawartość obu rejestrów (produkujących i konsumujących) jest identyczna, to oznacza, że kolejki są puste. Przykładowy stan rejestrów po odebraniu ramki pokazuje **rysunek 11**. Wszystkie kolejki zawierają po 4 elementy, toteż w rejestrze *MAC_RXDESCRIPTORNUM* jest wpisana liczba 3. Przed odebraniem ramki kolejki były puste, dlatego zawierały przykładowo liczbę 2 (*MAC_RXPRODUCEINDEX* i *MAC_RXCONSUMEINDEX*). Po odebraniu ramki zespół EMAC umieszcza jej zawartość w buforze określanym przez deskryptor o indeksie 2 i jej wielkość w odpowiednim miejscu w kolejce statusowej (również o indeksie 2) oraz zwiększa zawartość rejestru *MAC_RXPRODUCEINDEX* o jeden (po operacji zawiera 3). Różna zawartość rejestru „produkcyjnego" oraz „konsumenckiego" oznacza, że jest odebrana nowa ramka. Dane zawarte w buforze należy przepisać w inne miejsce i zwiększyć zawartość rejestru „konsumenckiego" o jeden, informując w ten sposób zespół EMAC, że dane zostały „sprzątnięte" z pamięci kolejkowej.



Rys. 11

Procedurę odbierania ramek pokazuje **listing 11**, gdzie następuje programowa realizacja wyżej opisanych czynności. Do pełnego zrozumienia jej działania wyjaśniam kilka szczegółów: instrukcja *EMACInstanceRec.InterfaceAllocFrameMemory* jest funkcją, która przydziela miejsce na odebraną ramkę gdzieś w pamięci operacyjnej, w pętli *for* realizowane jest przepisanie odebranej ramki z kolejki do przydzielonego obszaru (dla zwiększenia wydajności po cztery bajty), zwiększona jest zawartość rejestru konsumenckiego o jeden z uwzględnieniem ewentualnego „zawrócenia wskaźnika" na początek tablicy oraz interpretacji odebranych danych (wywołanie *EMACInstanceRec.InterfaceProcessFrame*).

Odnosząc działanie programu w zakresie dekapsulacji do historyjki Jasia Wędrowniczka, funkcja pokazana na listingu 11 odpowiada rozpakowaniu plecaczka, a wywołanie funkcji *EMACInstanceRec.InterfaceProcessFrame* prowadzi do funkcji pokazanej na **listingu 12** (wyjęciu woreczka), gdzie między innymi jest rozpoznany napis IP na woreczku (wariant instrukcji *case* o wartości *FRAME_IP*). To z kolei prowadzi do otwarcia woreczka i uzyskania pudełka. Mając napisane na pudełku hasło TCP, trafiamy poprzez wywołanie funkcji *IPLayerInstance->TCPProcessService* (**listing 13**) do funkcji pokazanej na **listingu 14**. Tam rozpoznany jest numer portu i poszukiwany jest otwarty port o odpowiednim numerze (wywołanie funkcji *LocateSocket_TCB* oraz *LocateSocket_Listen*).

Nadawanie ramek ethernetowych jest zbliżone do odbierania. Procedurę ilustruje **listing 15**, gdzie po przepisaniu zawartości nadawanej ramki do obszaru pamięci wskazanego przez odpowiedni deskryptor (odpowiedniego bufora w kolejkach związanych z EMAC), należy zwiększyć zawartość rejestru produkcyjnego związanego z funkcjonalnością nadawania. Tu program serwera jest „producentem" a zespół EMAC jest „konsumentem" danych. Podobnie jak przy odbieraniu, rejestr *MAC_TXPRODUCEINDEX* wskazuje na wolny deskryptor, który z kolei w polu *Packed* wskazuje na miejsce, do którego należy przepisać zawartość nadawanej ramki i po zakończeniu kopiowania zwiększyć zawartość rejestru (uwzględniając liczbę dostępnych deskryptorów), co z kolei jest sygnałem dla zespołu EMAC, by zabrać się do pracy.

W kolejnym odcinku będzie kontynuacja opisu działania programu serwera www.

W Elportalu w materiałach dodatkowych do tego numeru znajdują się następujące pliki:

- vircom_soft.zip – program źródłowy oraz binarny serwera www,
- serwercfg.zip – program źródłowy oraz binarny do konfiguracji serwera www,
- ramtest.zip – program źródłowy oraz binarny dla mikrokontrolera ARM realizującego test zewnętrznej pamięci RAM,
- bufory cykliczne.pdf – dokument opisujący ideę kolejek budowanych w oparciu o bufory cykliczne,
- ethernet_adresacja.pdf – dokument opisujący model OSI oraz ideę adresów IP oraz maski podsieci,
- ethernet_transmisja.pdf – dokument opisujący transmisję ramki ethernetowej.

Andrzej Pawluczuk
apawluczuk@vp.pl