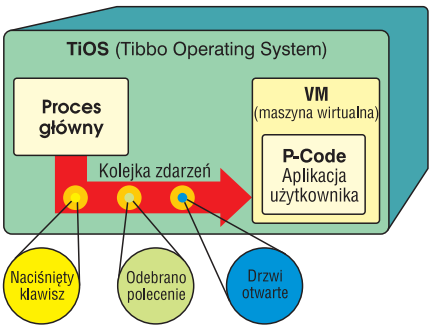
Ethernet w jednym module, część 1

Taiko: Ethernet w Basicu

Ethernet do niedawna był kojarzony jako wysoce wyspecjalizowane medium transmisji danych przeznaczone wyłącznie do budowy sieci komputerowych typu LAN. Jego zalety dostrzeżone przez producentów urządzeń automatyki przemysłowej takie jak: tania warstwa fizyczna, długi odcinek kanału bez konieczności stosowania repetera (huba lub switcha) oraz duże prędkości transmisji spowodowały, że zaczęli go również stosować producenci przyrządów pomiarowych. Ethernet pojawia się więc coraz częściej w ich wyrobach obok sprawdzonego standardu GPIB.

O zaszufladkowaniu Ethernetu jako skomplikowanego kanału transmisyjnego obsługiwanego jedynie przez duże systemy (komputery PC) możemy już chyba mówić w czasie przeszłym. W dobie nowoczesnych budynków z infrastrukturą sieci komputerowych, elastyczność, jaką daje Ethernet jest dużo większa, niż ta oferowana przez wyspecjalizowane magistrale. Popularyzacji Ethernetu sprzyja ponadto fakt, iż przeważnie zapewnia on również dostęp do Internetu. Czy można chcieć czegoś więcej? Chyba nie, ale należałoby się spodziewać, że ta uniwersalność i elastyczność jest jednak czymś okupiona. Intuicja nas nie myli, Ethernet to tak naprawdę zaledwie druga warstwa w modelu ISO/OSI, a wiec łącza danych. Jeśli



Rys. 1. Architektura systemu TiOS

chcielibyśmy używać choćby protokołu UDP (bądź – bardzo modnego sloganowo – TCP), musimy dołożyć jeszcze dwie warstwy, a jeśli chcielibyśmy skorzystać z HTTP (znów bardzo trendy...), to nagle okazuje się, że znajdujemy się na samym szycie modelu (warstwa aplikacji). Ogrom pracy, jaki należałoby włożyć w implementację poszczególnych warstw zabiłby wszelkie chęci konstruowania jakiegokolwiek urządzenia podłączanego do gniazdka RJ45. Na szczęcie nie jestem pierwszym, który zauważył ten problem. W chwili obecnej dostępnych jest kilka modułów typu OEM, można powiedzieć mostów Ethernet<–>RSxxx, które sprowadzają cały „problem" złożoności Ethernetu do... transmisji szeregowej. Od dłuższego czasu popularność zdobywają dwa takie moduły: MOXA–41xx, oraz Digi Connect Me. Ich największą zaletą jest to, że oprócz standardowej funkcji mostu oferują możliwość programowania. Dysponując tylko takim modułem jesteśmy w stanie oprzeć na nim cały projektowany system (posiadający interfejs Ethernet). Innymi znanymi czytelnikom EP modułami są produkty tajwańskiej firmy Tibbo. Charakteryzują się one solidnym wykonaniem, ale są też konkurencyjne cenowo. Do niedawna oferowały jedynie funkcjonalność mostu (brak możliwości uruchamiania własnych aplikacji), co dyskwalifikowało je w porównaniach z wyżej wspomnianymi modułami. Sytuacja ta zmieniła się z początkiem lutego, kiedy Tibbo zaprezentowało nowe rozwiązanie: Taiko.

Zupełnie inny

Taiko jest zestawem oprogramowania i rozwiązań umożliwiających rozwijanie aplikacji przez użytkownika, a następnie uruchamiania ich w modułach Tibbo (EM200, EM202 oraz ich pochodnych np. DS202). Zawiera:

– TiOS – system operacyjny, który uprzednio należy zainstalować w module (nadpisując oryginalny firmware),
– TIDE (*Tibbo Integrated Developement Environment*) – środowisko zawierające edytor kompilator i debugger,
– Tibbo BASIC – właściwy język programowania.

Po przeczytaniu informacji o zastosowanym języku programowania na myśl przychodzi oczywiście BASCOM. Widać, że użyty w module Taiko język ewidentnie stał się tutaj jedynie narzędziem. O ile BASCOM możemy zaliczyć do języków wysokiego poziomu, do czego upoważniają choćby zaimplementowanie w nim funkcje do obsługi peryferii (sam sposób pisania programu pozostał jednak tradycyjny), o tyle twórcy Taiko posunęli się krok dalej. Stworzyli pewien sposób programowania własnych modułów, który jest charakterystyczny dla aplikacji pisanych na komputery PC. Stworzono więc własny język używający elementów składni Basica. Identyczny efekt mógłby być osiągnięty również wtedy, gdyby język programowania mo-

dułów Taiko zdecydowano się oprzeć np. na języku C, tworząc wyimaginowane „Tibbo C". Wybór powyższej koncepcji był chyba doskonałym posunięciem z marketingowego punktu widzenia. Języki takie jak C i C++ chodź niesamowicie elastyczne i ciągle nowoczesne, a przy tym logiczne i nieprawdopodobnie uniwersalne, odstraszają! Uchodzą za trudne do nauczenia i wrogie dla programisty (oczywiście jest to nieprawda, ale żeby o tym się przekonać najpierw należy chcieć poznać C i koło się zamyka). Z Tibbo BASIC Taiko ma szansę na zgromadzenie zupełnie nowej rzeszy zwolenników zdolnych do pisania aplikacji, których działanie będzie co najmniej efektowne. Jednocześnie zawodowi programiści poznają Tibbo BASIC w ekspresowym tempie, a to za sprawą intuicyjnych mechanizmów, jakie twórcy Taiko zawarli w TIDE. Dodatkową zaletą jest to, że Taiko dostarcza komplet narzędzi do programowania i debugowania (wszystko poprzez Ethernet). Niebagatelny dla użytkowników jest również fakt, że Taiko jest darmowe. Oczywiście środowisko wykorzystujące Basic można uznać za „amatorskie" i niegodne uwagi, ale po historii z BASCOMEM byłaby to jedynie oznaka lekceważenia ogromnej liczby elektroników w Polsce. Można też powiedzieć, że Basic sam w sobie jest „powolny", co w klasycznym wydaniu tego języka jest prawdą, gdyż jest to język interpretowany. Taiko dostarcza jednak kompilator Tibbo Basic, co powoduje, że na platformie docelowej wykonywany jest kod binarny (podobnie zresztą jest w przypadku BASCOMA).

Jak to działa?

Oryginalny firmware w modułach Tibbo, a wiec ten odpowiedzialny za translację transmisji szeregowej do protokołu TCP czy UDP został wymieniony przez system operacyjny czasu rzeczywistego TiOS. System ten wykonuje tylko dwa procesy. Proces główny jest odpowiedzialny za komunikację z TIDE (jeśli jest konieczna) oraz za obsługę wszystkich peryferii systemu, w nomenklaturze Tibbo zwanych "obiektami platformy" (*Platform Objects*). Mamy, zatem do dyspozycji obiekty takie jak „ser" (*serial interface*) – odpowiedzialny za obsługę portu szeregowego, „sock" (*socket*) – inaczej gniazdo do ustanawiania połączeń TCP lub UDP, czy np. obiekt „net" – służący wprost do obsługi protokołu IP, a wiec do nadawania adresów IP, maski podsieci, adresu bramy.

Drugi z procesów TiOS to Maszyna Wirtualna (a wiec dostępny dla użytkownika programowy procesor) wykonująca właściwy kod uzyskany z kompilacji źródeł Tibbo BASIC. Twórcy Taiko nazwali go P–Code (pseudo kod), jako że wykonywany jest na niejako symulowanym procesorze (VM). Taki podział zadań w systemie zapewnia po pierwsze to, że nasz kod nie napotyka na żadne przestoje spowodowane np. opróżnianiem bufora nadawczego, czy zapisywaniem do wewnętrznej pamięci EEPROM – nad tymi zadaniami czuwa Proces Główny. Po drugie – nawet w przypadku złego kodu programu powodującego zablokowanie Maszyny Wirtualnej cały system pracuje dalej. Dzięki temu mamy np. ciągłą kontrolę nad modułem z poziomu TIDE. Architekturę TiOS przedstawiono na **rys. 1**. Innowacją (oczywiście tylko w świecie *embedded*) na poziomie samego sposobu pisania programu są w Taiko zdarzenia (*Events*), które wyznaczają przebieg wykonania programu.

Nie ma tu klasycznego liniowego wykonania kodu (jak to ma miejsce w ANSI C, czy choćby w Bascomie oraz w asemblerach). Cały program jest umieszczony w tzw. handlerach zdarzeń (można je porównać do funkcji obsługi przerwań), które są wywoływane po ich wystąpieniu. Proces główny monitorując peryferia sytemu (obiekty) tworzy kolejkę zdarzeń, która jest kierowana do maszyny wirtualnej. Mamy wiec np. *event* – „on_sys_init" generowany tuż po włączeniu modułu (posłuży nam on do inicjalizacji urządzenia), mamy *event* „on_ser_data_arrival" – generowany po odebraniu znaków przez port szeregowy (posłuży nam do odczytania bufora), itd. Lista zdarzeń (*Events*) oraz obiektów systemu zależy od zastosowanego modułu. Każdy obiekt ma tzw. właściwości (*properties*) i/oraz metody (odpowiednik funkcji w programowaniu liniowym). Przy pomocy właściwości ustawiamy i odczytujemy parametry obiektów (np. portu szeregowego). Przy pomocy metod uzyskujemy dostęp do ich funkcji (w sensie zadań, jakie mogą wykonać). Mamy, zatem np. metodę „ser.send" powodującą wysłanie znaków zawartych w buforze nadawczym portu szeregowego. Cała ta nomenklatura słusznie kojarzy się z programowaniem obiektowym tyle,

że nie jesteśmy w stanie definiować własnych klas obiektów, a nawet nowych egzemplarzy obiektów istniejących klas. Zatem pojęcie klasy tu nie występuje. Mamy jednie do czynienia z abstrahującym od sprzętu, programowym modelem modułu, który został za nas stworzony przez inżynierów Tibbo. Wyręczono nas w najtrudniejszej części programowania obiektowego, a więc w projektowaniu obiektowym.

Trzecim, ostatnim elementem Platformy, na której pracujemy są, tzw. funkcje systemowe (*syscalls*). Są on niczym innym, jak wbudowanymi w język funkcjami do obsługi stringów, konwersji liczb na kod ASCII itd.

Na sam koniec pozostawiłem moim zdaniem najbardziej przydatny składnik Taiko, a mianowicie wbudowany serwer HTTP. Pozwala on na dołączanie do projektu plików HTML, które następnie po prawidłowej konfiguracji socketów (ustawieniu ich w tryb HTTP) mogą być wyświetlane na żądanie przeglądarki www. Z poziomu plików HTML mamy dostęp do zmiennych naszego kodu i co ważniejsze, kod Basic może być umieszczany jako część kodu HTML pomiędzy znacznikami <?..?>. Zatem otrzymujemy język, który pozwala nam na umieszczanie kodu w treści strony www, który to kod będzie wykonywany na serwerze www Tibbo w momencie żądania strony. Rezultatem będzie zwrócenie prawidłowo sformatowanej strony www. Znów nasuwa się analogia do dużych systemów (serwer Apache, który interpretuje skrypty PHP, czy IIS z ASP). Oczywiście nic nie stoi na przeszkodzie, aby w pliku HTML zawrzeć również kod w postaci Java Script (z tym, że wykonają się one oczywiście po stronie klienta). W kolejnej części zaczniemy budowę projektu opartego na module EM202 – nie zdradzę na razie co to będzie.

Marcin Chruściel, EP
marcin.chrusciel@ep.com.pl