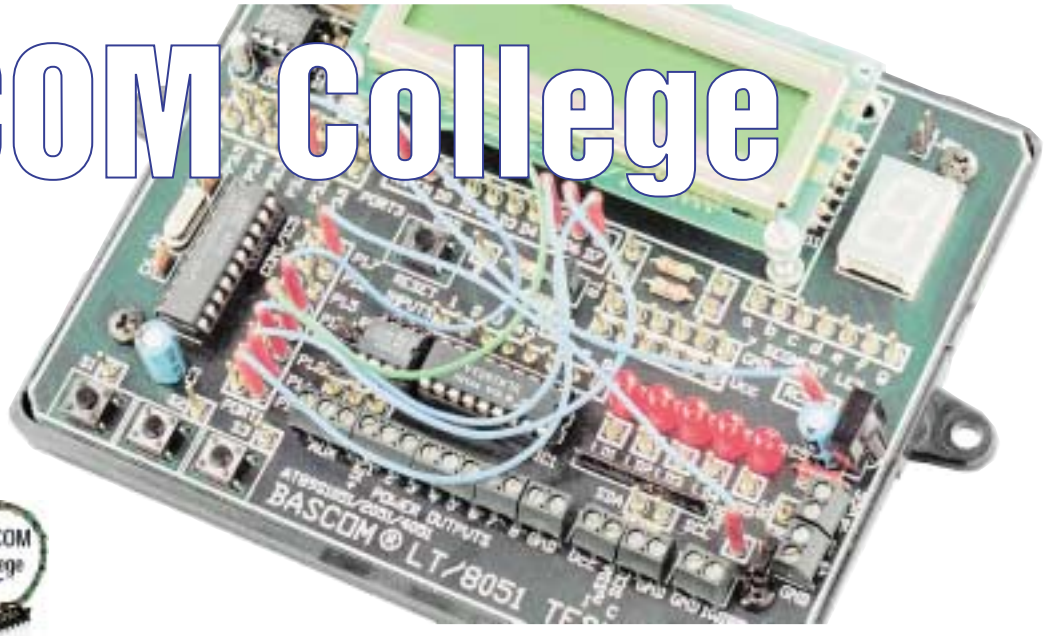
BASCOM College



Wykład 3

Do redagowania kolejnego wykładu BASCOM College przystępuję w zupełnie nowym, lepszym nastroju niż do pisania poprzednich artykułów! Nareszcie przestałem się poruszać po omacku i uzyskałem dane potrzebne mi do nadania kolejnym wykładom właściwej treści i formy. Przyczyniła się do tego analiza listów e-mailowych nadesłanych mi przez Czytelników. I tu od razu chciałbym przeprosić tych, którym nie zdążyłem indywidualnie odpowiedzieć na listy. Nie oznacza to bynajmniej, że zostały one wrzucone do kosza. Wprost przeciwnie, każdy list został przeczytany i przeanalizowany, a indywidualne odpowiedzi wysłane tylko tym Czytelnikom, którzy zadawali konkretne pytania. Jeżeli o kimś zapomniałem, to raz jeszcze przepraszam.

Ogólnie mówiąc, jestem więcej niż zadowolony. Oczywiście nie z siebie, ale z Was, Drodzy Studenci BASCOM College. W nadesłanych listach przeważały wypowiedzi osób może nie zawsze biegłych w sztuce programowania (w końcu, co mieliby oni do roboty w szkole), ale prawie zawsze MYŚLĄCYCH, co przy doskonaleniu umiejętności polegającej wyłącznie na konsekwentnym i popartym żelazną logiką myśleniu, ma kapitalne znaczenie. Nadesłane listingi nieraz aż roiły się od błędów, ale wynikały one najczęściej z pośpiechu i niecierpliwości oraz niezbyt dokładnej znajomości składni języka, a nie z błędów w rozumowaniu. Właśnie: niecierpliwość to chyba najczęstszy błąd, jaki popełniacie! Czego jak czego, ale cierpliwości programista musi mieć pod dostatkiem. Zawsze musicie pamiętać, że błąd może kryć się w każdej napisanej linijce, w jednym źle postawionym przecinku czy innym znaku. Oczywiście, kompilator wychwytuje i zgłasza wszelkie błędy w składni, ale nawet najmądrzejszy program komputerowy nie jest w stanie wykryć błędów w logice programu. Pisanie programów to przede wszystkim uwaga, cierpliwość, nieprzespane noce i hektolitry wypijanej podczas pracy szatańsko mocnej kawy. No i oczywiście "iskra Boża", bez tego też niewiele można zdziałać!

W kilku listach pojawiły się doniesienia o niesprawnym działaniu kompilatora, a nawet zarzuty o promowaniu nieudanego produktu. Niestety, po przeanalizowaniu fragmentów listingu zawsze okazywało się, że wina leżała po stronie niezbyt doświadczonego lub zadufanego we własne umiejętności programisty. Pakiet BASCOM używany jest przez tysiące elektroników na całym świecie i zdecydowana większość błędów została z niego dawno usunięta. Pamiętajcie zatem, że zanim zaczniecie podejrzewać kompilator o błędy, najpierw sprawdźcie sto razy, czy sami ich nie popełniacie. W 99 przypadkach na 100 "rację" ma kompilator, słusznie "czepiając" się niezrozumiałej dla niego składni języka.

Analizując Wasze listy e-mailowe doszedłem do wniosku, że największe problemy stwarzają Wam następujące tematy:

- Problemy z obsługą programatora, nieznajomość zasad jego konfigurowania.
- Koledzy, którzy zaczęli już pisać własne programy, na największe trudności napotkali podczas pisania tych ich fragmentów, które zajmują się obsługą timerów i przerwań. To oczywiste: znają oni najprawdopodobniej całkiem nieźle język BASIC, ale wyłącznie jego dialekty "komputerowe", w których obsługa timerów raczej nie występuje!
- Problemy z emulatorem sprzętowym, wynikające często z niezbyt dokładnego opisania jego działania w 3/00 Elektroniki dla Wszystkich. Cóż, mea culpa, jeżeli starczy nam czasu, to za chwilę poruszymy jeszcze ten temat.

A zatem te trzy wymienione sprawy będą tematem dzisiejszej lekcji. Pomęczymy się trochę z teorią, co nie oznacza, że nie wykonamy nowych, praktycznych ćwiczeń. Przy okazji omawiania posługiwania się timerami zaprojektujemy sobie pierwszy, wykonany w technice mikroprocesorowej, w pełni funkcjonalny zegarek.

Zanim jednak przejdziemy do konkretów, chciałbym wyczerpać do końca sprawę pomocy w Waszej pracy i rozwiązywania problemów, na jakie napotykacie. Nie jestem żadnym najwyższym autorytetem w tych sprawach (jak niesłusznie nazwał mnie jeden z Was). Najwyższym autorytetem jest mój Przyjaciel, pan Mark Alberts z Holandii, który o programowaniu procesorów **wie wszystko**. Zawsze możecie zwrócić się do niego o pomoc, oczywiście tylko w bardzo trudnych sprawach lub gdy jesteście prawie pewni, że znaleźliście pluskwę w programie. Możecie wysłać maila bezpośrednio do MCS Electronics, ale najlepszą metodą uzyskania pomocy i Marka, i setek ludzi z całego świata jest zapisanie się na listę dyskusyjną BASCOMMALIST (www.mcselec.com.) Rzecz jasna, korespondencja może być prowadzona wyłącznie w języku międzynarodowym, ale mam nadzieję, że na biurkach Kolegów nie władających do tej pory językiem amerykańskim leżą już stosowne podręczniki i chociażby bierne opanowanie tego języka jest już dla nich tylko kwestią czasu. Jeszcze raz zapraszam Was do dołączenia się do tej listy, jest wspaniała, tam nikt nie poszukuje cracków do kradzionego oprogramowania i nie zamieszcza ogłoszeń o "uruchamianiu" kodowanych odbiorników radiowych. Trwa tam nieustanna burza mózgów i wymiana doświadczeń, bez uwarunkowań typu "wiem, ale nikomu nie powiem", tak częstego na naszych listach.

Obsługa programatorów pakietu BASCOM8051

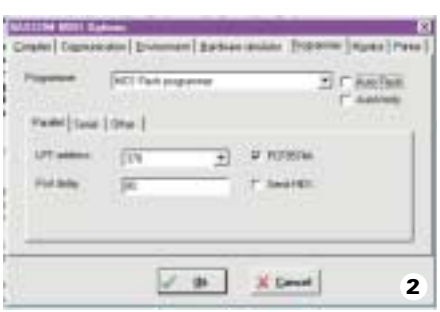
Fragment pakietu BASCOM, obsługujący programatory, jest jedną z najbardziej użytecznych i ciekawych części tego programu. Bez najmniejszej przesady mogę powiedzieć, że jest to "państwo w państwie", odrębne środowisko, które może być wykorzystywane łącznie z kompilatorem, ale także jako samodzielne oprogramowanie. Sądzę, że pełne opisanie tego fragmentu BASCOM-a jest obecnie sprawą najwyższej wagi, tym bardziej, że wiedza ta może być niezwykle użyteczna także dla Kolegów piszących programy w językach innych niż MCS BASIC. A więc, zaczynamy!

W poprzedniej lekcji napisałem, że najprostszą metodą posługiwania się programatorem jest zaznaczenie w menu OPTIONS PROGRAMMER opcji AUTO FLASH i po wybraniu typu programatora (najczęściej MCS Flashprogrammer, będący w Waszym posiadaniu) przechodzenie do programowania procesora za pomocą naciskania klawisza F4. Rzeczywiście, jest to metoda najprostsza, ale jak każe uproszczenie, nieco ograniczająca oferowane przez BASCOM możliwości. Przede wszystkim zamyka ona drogę do możliwości programowania procesorów za pomocą plików skompilowanych w innym niż BASCOM środowisku. Jednak nie tylko o to chodzi: prędzej czy później po naciśnięciu klawisza F4 pojawi się na ekranie komunikat o błędzie (**rys. 1**). I co wtedy?


1

Musimy zatem "dobrać się" do wszystkich, bogatych możliwości oferowanych nam przez obsługę programatora. Zakładam, że do komputera macie dołączony programator MCS Flashprogrammer opisany w Elektronice dla Wszystkich i że w podstawce programatora umieszczony jest jeden z procesorów rodziny 'X051. Nie ma przy tym znaczenia, czy jest to procesor "fabrycznie nowy", czy też zapisany jest już w nim jakiś program. Zakładam także, że napisaliście uprzednio i skompilowaliście jakiś program, przy czym także bez znaczenia jest jego treść.

Otwieramy zatem ponownie panel OPTIONS PROGRAMMER i tym razem "odznaczamy" opcję AUTO FLASH (**rys. 2**) i po zamknięciu tego okna naciskamy ponownie F4 ....


2

... Wow, ale bogactwo! Na ekranie pokazał nam się nowy, nieznany dotąd panel (**rys. 3**), a na nim cała masa przycisków i opcji. Zaczniemy od otwarcia okienka CHIP słusznie przypuszczając, że uzyskamy w ten sposób dostęp do operacji wykonywanych bezpośrednio na procesorze. Zanim jednak to uczynimy, wyjaśnijmy sobie, co właściwie znajduje się w największym oknie panelu. Otóż, w "białej" części środkowego panelu znajduje się treść skompilowanego uprzednio (lub, jak dowiemy się za chwilę, wczytanego) programu, zapisana w kodzie heksadecymalnym, a w części "niebieskiej" to samo, ale w kodzie binarnym.


3

Są to informacje, z których w obecnej chwili nie będziemy mieli większego pożytku, tak więc tylko przyjmujemy ich istnienie do wiadomości, a także zapamiętujemy że, w tym okienku możemy także dokonywać edycji programu w formacie HEX i naciskamy przycisk **CHIP**. Na ekranie pojawia się natychmiast kolejne okienko, a w nim następujące funkcje:

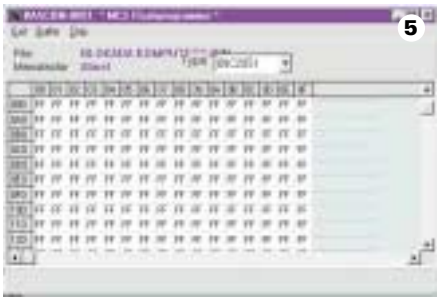
GET TYPE (określ typ procesora). Jest to bardzo pożyteczna funkcja, pozwalająca określić nie tylko typ procesora włożonego w podstawkę, lecz nawet jego producenta. Powyższe informacje "zaszyte" są wewnątrz każdego procesora, ale ponieważ nie mają żadnego wpływu na jego działanie, także tylko przyjmujemy ich istnienie do wiadomości. Czy jednak aby na pewno BASCOM jest taki mądry i potrafi jednoznacznie oznaczyć typ układu w podstawce programatora? Sprawdźmy to: zamknijmy na chwilę okienko CHIP i panelu głównym programatora zaznaczmy typ procesora inny, niż ten na którym dysponujemy (np. 89C1051). Ponownie otwieramy pasek CHIP i klikamy GET TYPE. Natychmiast powinna wyświetlić się informacja taka, jaką widzimy na rysunku 3, o ile oczywiście w podstawce znajduje się chip 89C2051. **Tu bardzo ważna uwaga: identyfikacja typu procesora jest zawsze (?) niezawodna, czego nie można powiedzieć o identyfikacji producenta. Niekiedy zdarza się, że program nie jest w stanie określić tej informacji i na ekranie pojawia się napis: MANUFACTURER UNKNOWN. Nie ma to najmniejszego znaczenia dla pracy programatora**, ale jest dość ciekawe. Kilkakrotnie napotykałem na to zjawisko wkładając w podstawkę nowiutkie procesory z napisem ATMEL, widocznym ze stu metrów, a program podawał, że nie może określić producenta chipa. Może podróbki?

ERASE (skasuj zawartość pamięci EEPROM procesora). Wykonanie tego polecenia spowoduje wymazanie **CAŁEJ** zawartości pamięci programu, **a także skasowanie bitów zabezpieczających**. Jest to bardzo ważne polecenie, a o jego użyteczności przekonamy się za chwilę.

SET LOCKBIT1 i SET LOCKBIT2 (ustaw bit zabezpieczający 1 i 2, **rys. 4**). Jest to także niezwykle ważna funkcja, za pomocą której w przyszłości będziemy mogli ... zabezpieczyć nie co innego, jak nasze interesy i korzyści finansowe, jakie będziemy mogli czerpać z naszej pracy.


4

Każdy programista jest wyłącznym właścicielem napisanego przez siebie programu i może nim dysponować w całkowicie dowolny sposób, także sprzedając dowolną liczbę jego kopii. Niestety, nie wszyscy, szczególnie w Polsce, rozumieją to prawo i respektują prawo autorskie. Ustawienie bitów zabezpieczających jest dokładnie tym samym, co założenie mocnych zamków w drzwiach domu: chroni nas przed, nazwijmy to po imieniu, złodziejami.


5

Kliknięcie na przycisk SET LOCKBIT 2 spowoduje, że zawartości pamięci programu nie będzie już można nigdy w żaden sposób odczytać. Nikt już teraz nie skopiuje Waszej pracy i nie "podzieli się" z Wami korzyściami z niej płynącymi. Natomiast kliknięcie SET LOCKBIT1 chroni Was nie tyle przed naruszeniem praw autorskich, ile raczej przed ... własnym roztargnieniem. Po ustawieniu tego bitu zabezpieczającego można wprawdzie odczytać zawartość pamięci procesora, ale nie można do niej niczego zapisać, chyba że pamięć EEPROM zostanie przedtem świadomie skasowana. Bardzo użyteczne, przekonałem się kilka razy! Czy jednak to co napisałem, znajduje odbicie w praktyce? Sprawdźmy zatem empirycznie działanie zabezpieczeń. Wybiegając w naszych rozważaniach nieco naprzód, otwórzmy panel BUFFER i kliknijmy przycisk PROGRAM CHIP. Dioda LED w programatorze zapali się na chwilę i zgaśnie po zaprogramowaniu procesora. Wróćmy teraz znowu do panelu CHIP i kliknijmy SET LOCKBIT 2. Tym razem dioda w programatorze tylko błyśnie i pozornie nic wielkiego się nie wydarzyło. Na pewno? Skoczmy zatem ponownie do panelu BUFFER i kliknijmy funkcję READ FROM CHIP (odczytaj zawartość pamięci programu procesora). Po kilku sekundach

w głównej części panelu pojawi się odczytana z pamięci procesora informacja (**rys. 5**). Jednak z pewnością nie jest to napisany przez nas i skompilowany program! Na ekranie pojawiły się same "FFki", co świadczy o tym, że pamięć procesora jest obecnie widziana jako czysta, nie zaprogramowana, i odczytywanie jej zawartości nie ma najmniejszego sensu.


6

Zajmijmy się teraz następnym panelem dostępnym z okienka programatora: BUFFER (rys. 6). W tym właśnie okienku znajdziemy najpotrzebniejsze w dalszej pracy narzędzia:

CLEAR (wyczyść bufor). Kliknięcie tej funkcji powoduje wyczyszczenie zawartości bufora programatora, która zostanie zapełniona samymi FFkami. Ta akurat funkcja jest stosunkowo mało użyteczna, ale jeżeli już jest...

READ FROM DISK (wczytaj dane z dysku do bufora). **To chyba jedna z najważniejszych opcji programatora! Umożliwia ona wczytanie do bufora, a następnie zaprogramowanie procesora dowolnym programem na 'X051 (a także kilka innych typów procesorów, m. in. na 8052), zapisanym w kodzie binarnym, czyli praktycznie każdego pliku binarnego wygenerowanego przez dowolny kompilator obsługujący dowolny język programowania. Nie obowiązują tu żadne ograniczenia wersji DEMO: wczytywany plik binarny może mieć dowolną wielkość, oczywiście mieszczącą się w wybranym typie procesora**.

Dane do bufora programatora wczytywane są zgodnie z ogólnie przyjętymi w systemie WINDOWS zasadami, możemy przechodzić z jednego katalogu do drugiego, a wyświetlane są wyłącznie pliki typu ***.BIN** (**rys.7**).


7

WRITE TO DISK (zapisz zawartość bufora programatora na dysku). Ta opcja pozwala na zapisanie na dysku, w dowolnym katalogu i pod dowolną nazwą pliku znajdującego się aktualnie w buforze programatora. Może mieć ona zastosowanie przy zapisywaniu plików odczytanych z pamięci procesora poleceniem:

READ FROM CHIP (odczytaj plik z pamięci EEPROM procesora). Kliknięcie tego przycisku spowoduje wczytanie do bufora programatora zawartości pamięci programu procesora, o ile ta nie została zabezpieczona lock bitami i posiadamy na to zgodę autora programu. Odczytanymi danymi możemy następnie zaprogramować dowolną liczbę procesorów (widzimy tu właśnie sens stosowania zabezpieczeń) lub zapisać je na dysku.

WRITE TO CHIP (zapisz zawartość bufora w pamięci procesora). Wydanie tego polecenia spowoduje przeniesienie zawartości bufora do pamięci EEPROM procesora. **Uwaga: wydając to i tylko to polecenie można wpuścić się w niezłe maliny i należy stosować je z dużą ostrożnością. Programator po wydaniu tego polecenia kopiuje zawartość bufora do procesora, nie zwracając uwagi, czy w pamięci programu nie pozostały jakieś zbędne dane!** Nic jednak tak nie przekonuje, jak celowy i sprawnie przeprowadzony eksperyment:

a) Zaprogramujmy procesor jakimkolwiek programem, najlepiej o znacznej objętości. Możemy tu wykorzystać przycisk PROGRAM CHIP, którego działanie omówimy za chwilę.

b) Napiszmy sobie dowolny krótki programik, byle co, może robić np. " LCD "TEST" " i skompilujmy go.

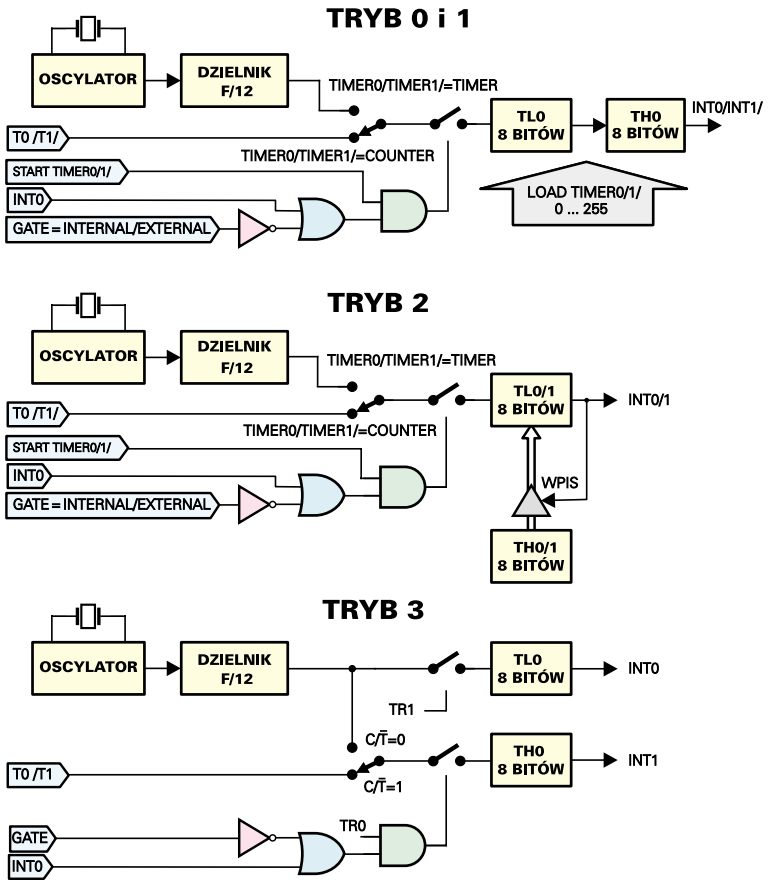
c) Wejdźmy z powrotem do panelu programatora, wybierzmy CHIP i WRITE TO CHIP. Zapalenie się diody w programatorze mogłoby świadczyć, że operacja programowania przebiegła pomyślnie. Nie radzę jednak wkładać tak zaprogramowanego procesora w podstawkę płytki testowej i sprawdzać jego działania. Powróćmy zatem do opisu panelu BUFFER, aby dowiedzieć się, czy procesor został na pewno zaprogramowany poprawnie!

VERIFY (sprawdź zgodność zawartości pamięci bufora z zawartością pamięci EEPROM procesora). Przycisk ten pozwala na weryfikację poprawności programowania i po jego naciśnięciu z pewnością pojawi się przed naszymi oczami napis taki, jak na **rysunku 8**. Co to oznacza, dlaczego programowanie procesora się nie udało? To proste: przed programowaniem z pamięci procesora nie został usunięty poprzedni program i w EEPROM-ie powstał przysłowiowy "groch z kapustą"!


8

Programowanie pamięci EEPROM mikroprocesorów 'X051 możliwe jest tylko wtedy, kiedy każdy bajt, który ma zostać zaprogramowany ma wartość FF. W praktyce oznacza to, że pamięć używanego już procesora musi zostać skasowana przed rozpoczęciem ponownego programowania. Zostawmy zatem polecenie WRITE TO CHIP do celów specjalnych i zwykle używajmy:

PROGRAM CHIP (zaprogramuj procesor). Wydanie tego polecenia spowoduje wykonanie przez programator dwóch kolejno następujących po sobie czynności: skasowania pamięci procesora i zapełnienie jej samymi FFkami, a następnie wpisanie do tej pamięci nowego programu. Sprawdźmy działanie tego polecenia, naciskając kolejno PROGRAM CHIP, a po zaprogramowaniu procesora: VERIFY. Tym razem wyświetlony w dolnej części panelu komunikat **"Buffer and chip are identical"**



9

(bufor i zawartość pamięci procesora są identyczne) oznajmił nam o poprawnym przeprowadzeniu operacji programowania procesora.

W panelu programatora pozostał nam już tylko jeden przycisk do naciśnięcia: EXIT. Zróbmy to zatem, a działanie tego ostatniego polecenia pozwoli nam, po chwili odpoczynku i wypiciu filiżanki dobrej kawy, zabrać się za kolejny temat:

Timery

Bloki timerów - liczników umieszczone wewnątrz struktury procesorów '51 są jednymi z najbardziej użytecznych układów, jakie możemy wykorzystywać w naszej pracy. Umożliwiają one dokonywanie wszelkich operacji rozgrywających się w czasie rzeczywistym, budowanie zegarów, timerów, stoperów oraz innych układów, których działanie uzależnione jest od upływu czasu. Jednocześnie, obsługa timerów sprawia zwykle wiele problemów początkującym programistom. Utarło się nawet przekonanie, że operacje związane z timerami i przerwaniami to przysłowiowa "czarna magia" i że lepiej się za nią nie zabierać. Cóż, przy odrobinie uwagi poradzimy sobie z pewnością tak z timerami, jak i na następnej lekcji z przerwaniami, a "zabierać się" za te tematy musimy, bo inaczej nie będziemy w stanie napisać większości potrzebnych nam programów. Mam zresztą zamiar udowodnić Wam, że wykorzystywanie timerów jest naprawdę banalnie proste (no chyba, że używamy kilku timerów i przerwań jednocześnie, wtedy trzeba się trochę pomęczyć). Dowód przeprowadzimy w najbardziej spektakularny sposób: zbudujemy wspólnie nasz pierwszy zegar mikroprocesorowy: prosty, lecz w pełni funkcjonalny układ czasomierza, mogący znaleźć liczne zastosowania praktyczne. Najpierw jednak zapoznajmy się z niezbędnymi informacjami teoretycznymi.

Procesory podrodziny 'X051 wyposażone są w dwa umieszczone w ich strukturze 2 x 8-bitowe timery: Timer0 i Timer1. Timery możemy traktować trochę jako osobne układy, jedynie sterowane z procesora i potrafiące zgłaszać mikrokontrolerowi fakt przepełnienia ich zawartości. Do rejestru timera możemy w każdej chwili zapisać dowolną liczbę (oczywiście, z dozwolonego aktualnie stosownym trybem zakresu), zatrzymać lub uruchomić liczniki. Impulsy zliczane przez timery mogą być równie dobrze pobierane z wnętrza procesora, jak i z zewnętrznego źródła. Możliwości jest wiele i najwyższy czas trochę uporządkować nasze wiadomości na temat timerów - liczników.

Każdy timer składa się z dwóch rejestrów - liczników TL[numer timera] i TH[numer timera]. Timer może pracować w jednym z czterech, określonych programowo trybów:

Tryb 0

Tryb pracy 0 jest identyczny dla licznika TO i licznika TL. W trybie tym układ pracuje jako 13-bitowy czasomierz / licznik. Stan licznika określany jest zawartością rejestru TH[0,1] oraz pięciu młodszych bitów rejestru TL[0,1]. Stan trzech starszych bitów rejestru TL[0,1] jest nieokreślony. Przepełnienie licznika, czyli zliczenie impulsu, wskutek którego stan wszystkich bitów zmienia się z 1 na 0. Przepełnienie licznika może być sygnalizowane zgłoszeniem przerwania TIMER[0,1].

Zliczanie ma miejsce tylko wtedy, gdy wydane zostało na to zezwolenie za pomocą plecenia START TIMER[0,1]. I może być zatrzymane poleceniem STOP TIMER[0,1].

Tryb 1

Tryb 1 pracy liczników TO i T1 jest niemal identyczny z trybem 0. Jedyną różnicą jest wykorzystywanie pełnej pojemności rejestrów TL[0,1], a tym samym zwiększenie pojemności licznika z 13 do 16 bitów.

Tryb 2

W trybie 2 liczniki pracują jako 8-bitowe liczniki (TL[0,1]), z automatycznym przeładowaniem. Przepełnienie TL[0,1] nie tylko powoduje wygenerowanie przerwania, ale jednocześnie ładuje TL[0,1] zawartością TH[0,1]. Zawartość TH[0,1] może być w dowolnej chwili zmieniana programowo. Operacja przeładowania nie powoduje zmiany zawartości rejestru TH[0,1].

Tryb 3

Tryb 3 jest jedynym trybem, w którym sposób działania licznika T1 różni się od licznika T0. Wprowadzenie licznika T1 w tryb 3 pracy powoduje jego zatrzymanie, czyli daje ten sam efekt, co wydanie polecenia STOP TIMER1 w którymkolwiek z pozostałych trybów pracy licznika T1. Licznik T0 w trybie 3 wykorzystuje swoje obydwie połówki (THO i TLO) jako dwa niezależne liczniki 8-bitowe. Jeden z nich (utworzony z TLO) funkcjonuje, podobnie jak cały licznik T0, w trybie 0 lub 1, przy czym jedyną różnicą jest pojemność licznika wynosząca 8 bitów. Druga część (THO) tworzy 8-bitowy czasomierz zliczający cykle maszynowe. Włączanie i wyłączanie czasomierza odbywa się za pomocą polecenia START TIMER0, natomiast jego przepełnienie powoduje wygenerowanie przerwania TIMER0.

Zanim rozpoczniemy jakiekolwiek doświadczenia z timerami, zbierzmy razem wszystkie polecenia języka MCS BASIC, służące ich sterowaniu, i wyjaśnijmy sobie ich znaczenie. Najważniejszym poleceniem, jakie należy wydać przed rozpoczęciem pracy timerów jest określenie ich konfiguracji.

Ogólna postać tego polecenia wygląda następująco:

CONFIG TIMER0= COUNTER/TIMER, GATE=INTERNAL/EXTERNAL, MODE=0-3

Musimy zatem określić wstępnie trzy najważniejsze parametry decydujące o sposobie pracy każdego z timerów.

1. **TIMER[0,1] = TIMER**
- wskazany timer zlicza impulsy pochodzące z wewnętrznego oscylatora

TIMER[0,1] = COUNTER
- wskazany timer zlicza impulsy dostarczane na wejście T0 lub T1

2. **GATE = EXTERNAL** - timer, do którego odnosi się to polecenie będzie uruchamiany i zatrzymywany stanem na wejściu przerwania INT0 lub INT1

GATE = INTERNAL - timer, do którego odnosi się to polecenie będzie uruchamiany i zatrzymywany poleceniami programowymi

3. **MODE** - określenie trybu pracy wskazanego timera (omówione powyżej)

Każdy z timerów, niezależnie od wybranego trybu pracy, może być wstępnie "załadowany" wartością, od której po uruchomieniu rozpocznie zliczanie. Ładowania timera dokonujemy za pomocą następujących poleceń programowych:

LOAD TIMER[0,1], [wartość]
gdzie wartość jest liczbą z przedziału 0-255 dla trybu 2, lub 0 - 65535 dla trybu 1
lub
COUNTER[0,1] = [wartość]
gdzie wartość jest liczbą z przedziału 0-255 dla trybu 2, lub 0 - 65535 dla trybu 1

Tu bardzo ważna uwaga:
Timer zlicza impulsy od zadanej (załadowanej) liczby w górę, aż do osiągnięcia stanu 11111111 (w trybie2) lub 11111111 11111111 (w trybie 1). A zatem, aby otrzymać sygnał przerwania np. po 10 impulsach, należałoby do timera załadować wartość równą 256 - 10 czyli 246. Na szczęście język MCS BASIC zwalnia nas od konieczności dokonywania nawet tak prostego obliczenia (odnoszę czasami wrażenie, że dewizą Marka jest "Co tu wymyślić, aby inni nie musieli myśleć") i sam wykonuje potrzebne kalkulacje.
A zatem, wydanie polecenia:

LOAD TIMER0, 10

Spowoduje wystąpienie przerwania po 10 cyklach maszynowych procesora.

Następujące polecenia służą uruchamianiu i zatrzymywaniu timerów:

START TIMER[0,1] - włącza wskazany timer. Równoważne mu jest polecenie:
START COUNTER[0,1]

STOP TIMER[0,1] - zatrzymuje wskazany timer. Równoważne mu jest polecenie:
STOP COUNTER[0,1]

Trudno sobie wyobrazić sytuację, w której uruchomiony timer działałby sobie, a muzom, nie wywierając jakiegokolwiek wpływu na program sterujący pracą procesora. Musimy zatem poinstruować procesor, jakie czynności mają zostać wykonane w przypadku wystąpienia przerwania (przepełnienia) pracującego timera. Służy temu polecenie programowe:

ON TIMER[0,1] [nazwa podprogramu]

Gdzie podprogram powinien mieć następującą postać:

Nazwa podprogramu:
Czynności do wykonania
...
...
...
RETURN

Bardzo często zdarza się, że nie mamy zamiaru wykorzystywać przerwań przez cały czas pracy programu. Dlatego też na obsługę przerwań muszą zostać wydane specjalne zezwolenia, które w każdym momencie mogą zostać cofnięte.

ENABLE INTERRUPTS - globalne zezwolenie na obsługiwanie przerwań

ENABLE TIMER[0,1] - zezwolenie na obsługę przerwania wskazanego timera
i analogicznie:

DISABLE INTERRUPTS - globalny zakaz obsługi przerwań

DISABLE TIMER[0,1] - zakaz obsługi przerwania wskazanego timera

Przerwania, ich obsługa i problemy wynikające z ich stosowania będą tematem jednego z następnych wykładów. Zagadnieniu temu poświęcimy szczególnie dużo czasu, ponieważ umiejętne stosowanie przerwań jest problemem nie tylko dla początkujących programistów. Wyjątkowo trafną i precyzyjną definicję przerwania zamieścił we wspomnianej na początku tego artykułu książce pan Tomasz Starecki:

"Idea przerwań polega na tym, że w odpowiedzi na określony sygnał (sygnał przerwania) mikrokontroler zawiesza chwilowo wykonywanie programu głównego i wykonuje specjalną procedurę określaną mianem procedury obsługi przerwania. Po zakończeniu tej procedury mikrokontroler wraca do wykonywania programu głównego, począwszy od miejsca, w którym zostało ono zawieszone.

Podstawową funkcją przerwań jest umożliwienie szybkiego reagowania na wydarzenia zewnętrzne. Zaletą wykorzystywania przerwań jest też uproszczenie oprogramowania. Wynika ono m.in. stąd, że program główny jest uwalniany od konieczności testowania stanu wybranych układów peryferyjnych lub linii sygnałowych, ponieważ w razie stosowania przerwań operacje te są realizowane sprzętowo.

```
Config Timer0 = Timer , Gate = Internal , Mode = 2      'konfiguracja
timera 0
Load Timer0 , 5                                         'załadowanie do
timera wartości 256-5
On Timer0 Timerinterrupt                                'po wystąpieniu
przerwania skocz do podprogramu timerinterrupt
Enable Timer0                                           'zezwalam na
obsługę przerwania z timera 0
Enable Interrupts                                       'globalne zezwolenie na obsługę przerwań
Start Timer0                                            'start timera 0

Do                                                      'program w pętli
oczekuje na przerwanie
Loop

Timerinterrupt:                                         'podprogram obsługi przerwania
Print "Timer interrupt!"                                'po wystąpieniu przerwania napisz ten tekst
End                                                     'koniec programu
Return                                                  'dodane
"dla porządku", po poleceniu END nie ma właściwie sensu
```

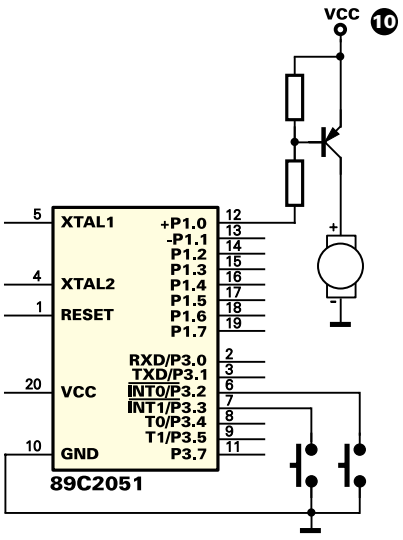
W niektórych mikroprocesorach jedynym rodzajem przerwań są przerwania zewnętrzne. W mikrokontrolerach, oprócz przerwań zewnętrznych, występują też z reguły przerwania od wewnętrznych układów peryferyjnych (np. timerów, dod. Z.R.)."

Na obecnym etapie nauki ta definicja powinna nam wystarczyć.

Najwyższy czas przejść od teorii do praktyki i wypróbujmy działanie timerów w konkretnych przykładach. Posłużymy się w tym celu emulacją programową, której główna wada - ślimacze tempo pracy, stanie się w tym momencie zaletą. Operacje wykonywane "w rzeczywistym świecie" przez timery rozgrywają się w takim tempie, że obserwacja ich byłaby praktycznie niemożliwa. Napiszmy sobie zatem następujący, prosty programik:

Skompilujmy go i uruchommy w symulacji programowej. Jeszcze raz przypominam, że podawane przykłady odnoszą się do trybu „symulacji procesora" i „żywcem" przeniesione do „rzeczywistego układu" nie będą pracować poprawnie. Po naciśnięciu przycisku START w symulatorze prawie natychmiast w okienku emulatora terminala pojawił się napis "Timer interrupt!". Chyba wszystko jest OK, ale na wszelki wypadek sprawdźmy jeszcze raz, zmieniając wartość czasu oczekiwania na np. 250. Po powtórnym skompilowaniu i uruchomieniu w symulacji okazało się, że tym razem czekaliśmy na wystąpienie przerwania znacznie dłużej! Bingo, eksperyment potwierdził przewidywania teoretyczne! Jeżeli jednak to doświadczenie się udało, to przeprowadźmy jeszcze kilka innych, także w symulacji programowej.

Usuńmy teraz polecenie END kończące działanie programu po wystąpieniu przerwania i uruchommy nasz programik jeszcze raz. Oczywiście, nie zapominajmy o kompilowaniu go po wprowadzeniu każdej zmiany! Okazało się, że tym razem program nie zatrzymuje się, ale po każdym przerwaniu wysyła na ekran komunikat o jego wystąpieniu. Ciekawe, to może mieć już jakieś zastosowanie praktyczne! Wiemy przecież, że oscylator kwarcowy procesora charakteryzuje się dość dobrą stabilnością i to "kółko", które utworzyliśmy, będzie "kręcić" się z w miarę stałą prędkością, niezależnie od innych czynności wykonywanych przez program. Dopiszmy zatem jeszcze kilka linijek do podprogramu obsługi przerwania, który po tych przeróbkach będzie miał następującą postać:



Uruchommy ten kolejny programik eksperymentalny. No i co najlepszego zrobiliśmy? Na wyświetlaczu ukazują się liczby od 0 do 59, dwie diody symbolizujące stany dwóch pinów portu P3 zapalają się i gasną co pewien czas ... Pewien? Przecież jedna dioda miga mniej więcej co sekundę, a druga co prawie minutę! A zatem, o tak mimochodem, zbudowaliśmy metodami programowymi nic innego, jak sekundnik do zegara mikroprocesorowego! Oczywiście, tak napisany program może działać w miarę poprawnie tylko w symulacji. W "real world" po zaprogramowaniu procesora tym programem, tak zbudowany sekundnik nie miałby najmniejszej szansy na poprawne działanie. Owszem, może by działał, ale nawet przy zastosowaniu "najwolniejszego" kwarcu z taką szybkością, że procesor nawet nie nadążyłby wyświetlać danych, a my ich zaobserwować. Z programem mikroprocesorowego zegarka zapoznacie się znacznie szybciej, niż być może wielu z Was przypuszcza. I w dodatku będzie to dość bajerancki zegarek, z kilkoma funkcjami jak dotąd nie stosowanymi w tego typu urządzeniach. Zanim jednak to nastąpi, poćwiczmy jeszcze trochę używanie timerów i spróbujmy wykonać, na razie tylko w symulacji, kolejne "urządzenie użytkowe".

Problem jest następujący: zadaniem budowanego układu mikroprocesorowego będzie np. regulacja prędkości obrotowej silnika prądu stałego lub zmiana wartości prądu płynącego w obwodzie ładowarki do akumulatorów. Można te problemy rozwiązać metodami klasycznymi - analogowymi, licząc się z mocą strat odkładającą się na elementach regulacyjnych. Nie po to jednak uczymy się nowoczesnej techniki mikroprocesorowej, aby cofać się teraz w przeszłość. Zastosujemy regulację PWM (Pulse Width Modulation).

```
Dim Seconds As Byte 'deklaracja zmiennej SECONDS jako liczby z zakresu 0 ... 255
Config Lcd = 16 * 1         'konfiguracja wyświetlacza LCD w symulacji
Config Timer1 = Timer , Gate = Internal , Mode = 2
Load Timer1 , 5
On Timer1 Timerinterrupt
Enable Timer1
Enable Interrupts
Start Timer1

Do

Loop

Timerinterrupt:
Set P3.1
Reset P3.1
Incr Seconds                'zwiększ wartość zmiennej SECONDS o 1

If Seconds = 60 Then        'jeżeli wartość zmiennej SECONDS
jest równa 60 to:
Seconds = 0                 'zmień wartość zmiennej SECONDS na 0
Set P3.2
Reset P3.2
End If                      'koniec uwarunkowania

If Seconds < 10 Then        'jeżeli wartość zmiennej SECONDS jest mniejsza od 0 to:
Lcd "0"                     'wyświetl cyfrę 0 na wyświetlaczu LCD
End If                      'koniec uwarunkowania
Lcd Seconds                 'wyświetl wartość zmiennej SECONDS
Home                        'ustaw kursor na początku wiersza
Return                      'koniec podprogramu
```

Na **rysunku 10** pokazany został fragment układu, który zamierzamy budować. Tranzystor wykonawczy dołączony jest do pinu P1.0 procesora. Nie zastanawiajmy się na razie, jakie czynniki będą wpływały na szerokość impulsów pojawiających się na wyjściu procesora. Dla uproszczenia ćwiczenia przyjmijmy, że wzrost szerokości impulsów, a tym samym **zmniejszenie** (tranzystor T1 przewodzi przy stanie niskim na P1.0!) mocy oddawanej do obciążenia powodowane będzie podaniem stanu niskiego na wejście INT0, a zmniejszenie wypełnienia PWM - podaniem stanu niskiego na wejście INT1.

Napiszmy sobie teraz malutki programik, którego listing przedstawiony jest poniżej. Ponieważ jest to już ostatnie zadanie stojące

```
Dim Pwm1 As Byte
Dim Pwm2 As Byte
Pwm1 = 50
Pwm2 = 100 -  Pwm1
Config Timer1 = Timer , Gate = Internal , Mode = 2
Load Timer1 , Pwm1
On Timer1 Timerinterrupt
Enable Timer1
Enable Interrupts
Enable Int0
Enable Int1
On Int0 Incrpwm
On Int1 Decrpwm
Start Timer1

Do
Loop

Timerinterrupt:
P1.0 = Not P1.0
P1.1 = Not P1.0
If P1.0 = 1 Then
                        Load Timer1 , Pwm2
                        Else
                        Load Timer1 , Pwm1
                    End If
Return

Incrpwm:
Incr Pwm1
                    If Pwm1 = 96 Then
                        Pwm1 = 95
                    End If
Pwm2 = 100 -  Pwm1
Print Pwm1 : Print Pwm2
Return

Decrpwm:
Decr Pwm1
                    If Pwm1 = 4 Then
                        Pwm1 = 5
                    End If
Pwm2 = 100 -  Pwm1
Print Pwm1 : Print Pwm2
Return
```

Z ostatniej chwili

1. Zakończone zostało testowanie przez Marka Albertsa nowego emulatora sprzętowego, opracowanego wspólnie przez MCS Electronics i Elektronikę Praktyczną. Opis tego układu zostanie zamieszczony być może już w najbliższym numerze EP.

3. A teraz jeszcze jedna najważniejsza sprawa: chciałbym polecić źródło dodatkowej wiedzy o procesorach rodziny '51. Jest oczywiste, że w ramach wykładów prowadzonych w BASCOM College nie będziemy w stanie szczegółowo omówić wszystkich procesorów tej rodziny. Z konieczności nawet nasz podstawowy procesor, AT89C2051 zostanie opisany skrótowo, bez wgłębiania się w zawiłe tajniki jego budowy wewnętrznej. Przyjęliśmy zresztą w naszym College zasadę "czarnej skrzynki", która wykonuje wydane jej w języku MCS polecenia, ale mało nas obchodzi, jak to robi. Sądzę jednak, ze wielu Czytelników chciałoby pogłębić swoją wiedzę i skorzystać z dobrej literatury uzupełniającej. Taką literaturą jest książka "Mikrokontrolery jednoukładowe rodziny 51" autorstwa pana Tomasza Stareckiego. Sam wiele nauczyłem się z tej książki i towarzyszy mi ona nieustannie podczas przygotowywania wykładów BASCOM College. Książka ta znajduje się w ofercie Księgarni Wydawniczej.

przed nami w dniu dzisiejszym, pozostawię ten program bez komentarzy. Dodam tylko, że tak napisany program może służyć wyłącznie celom szkoleniowym i pracy w symulacji programowej. Jest to ćwiczenie teoretyczne, a w jaki sposób należy stosować przerwania zewnętrzne (INT0 i INT1) w praktycznych układach, dowiemy się na następnych lekcjach.

Nasz program testowy należy skompilować i uruchomić w emulacji programowej. Efekt zwiększania i zmniejszania wypełnienia impulsów uzyskamy naciskając przyciski INT0 i INT1 w okienku symulatora, a obserwować go możemy patrząc na czerwone diody symbolizujące stany portów procesora (**rys.11**).


11

Niestety, GAME OVER i nie zdążymy już dzisiaj poruszyć ważnego tematu emulatorów programowego i sprzętowego. Będzie to zatem główny temat najbliższej lekcji, na której zapoznamy się także z obsługą przerwań zewnętrznych.

Zbigniew Raabe
e-mail: zbigniew.raabe@edw.com.pl

BASCOM College 3 - suplement

No i doigrałem się. Po AVT rozeszła się wieść, że nasz Szef żąda mojej głowy, podanej mu na srebrnej tacy. Ponieważ dziwnie zależy mi na tej właśnie głowie i nie miałem ochoty podzielić smutnego losu Jana Chrzciciela, natychmiast zgłosiłem się „na dywanik" i dowiedziałem się, co następuje:

„Panie Zbyszku, niech pan wreszcie trochę uporządkuje ten „kontrolowany" bałagan w BASCOM College! Przecież najwyższy już czas, aby Studenci otrzymali podstawowe informacje o programie zajęć, ile czasu ma trwać nauka i jakie będą tematy najbliższej lekcji! Nie wszystko można przecież puścić „na żywioł" , nawet w największym szaleństwie musi być jakaś metoda."

Co racja, to racja, trzeba rzeczywiście na chwilę skupić się na tych ważnych sprawach i ogłosić to, co powinno być ogłoszone na samym początku:

Plan zajęć BASCOM College

Niestety, precyzyjne określenie tematu każdej następnej lekcji programu BASCOM College nie jest takie proste. Mamy do czynienia z dwoma, nieprawdopodobnie gwałtownie rozwijającymi się żywiołami: elektroniką i oprogramowaniem. Nie mam najmniejszego pojęcia, jaką „rakietę" wystrzeli w najbliższym czasie Mark Alberts z MCS Electronics, jakie nowe funkcje zostaną dodane do BASCOM'a. Nie mam także pojęcia, jakie nowe procesory ukażą się na rynku i jaka będzie na to reakcja twórców oprogramowania. No i najważniejsze: nie wiem też na jakie problemy napotkacie w czasie nauki i jakie dodatkowe tematy zechcecie poruszyć na kolejnych wykładach. Dlatego też jestem w obecnej chwili w stanie określić jedynie główne tematy następnych lekcji, a tematy uzupełniające niech pozostaną na razie wielką niewiadomą.

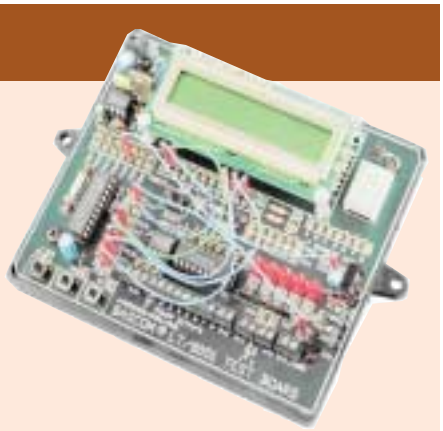
Zanim jednak przejdziemy do szczegółów, odpowiedzmy najpierw na często zadawane przez Was pytanie: jak długo ma potrwać nauka w BASCOM College? Sądzę, że podstawowy materiał powinniśmy przerobić najdalej do końca tego roku, czyli zarazem do końca XX stulecia. A więc w nowe stulecie wejdziemy już jako „pasowani" elektronicy, posiadając umiejętność programowania i budowania układów elektronicznych wykorzystujących procesory '51. Celowo napisałem '51 a nie 'X051, bo na kolejnych wykładach zajmiemy się także „dużymi pięćdziesiąt jedynkami" i otrzymacie także niezbędny do ich obsługi hardware. A co dalej? Z pewnością działalność BASCOM College jeszcze się nie zakończy, zakończy się tylko podstawowy jego temat jakimi są procesory '51. Z pewnością zajmiemy się też „młodszym bratem" BASACOM'a8051 - BASCOM'em AVR!

Procesory '51 są pewnego rodzaju standardem i pozostaną nim jeszcze przez długie, długie lata. Nie możemy jednak nie zauważać innych rodzin procesorów, wśród których rewelacyjnie szybkie i wyposażone w wiele nowoczesnych funkcji procesory AVR wysuwają się z pewnością na pierwszy plan. Na szczęście, nauka programowania tych układów będzie już bardzo prosta, ponieważ z poziomu pakietu BASCOM AVR praktycznie nie różni się od .... programowania procesorów '51. Nie zajmie nam więc to więcej niż dwa, trzy wykłady. Co dalej? Nie wiem, ale mam pewne plany, pewne marzenia....

Mark Alberts wspominał coś na temat kompilatora procesorów AVR opartego na modyfikowanym dialekcie języka C+. Otoczenie miałoby być identyczne ze środowiskiem BASCOM'ów, tylko język inny ... Może tym się zajmiemy? A może uda mi się namówić Marka do stworzenia nowego kompilatora, tym razem procesorów ST? Pożyjemy, zobaczymy, ale już w przyszłym wieku!

Jedno jednak jest pewne: po zakończeniu programu BASCOM College postaramy się stworzyć, być może wspólnie z redakcją Elektroniki Praktycznej i MCS Electronics elitarny klub, będący w pewnym sensie kontynuacją BASCOM College. Będzie to międzynarodowe forum służące wymianie doświadczeń, zapoznawaniu jego Członków z nowościami i nawiązaniu kontaktów, za pośrednictwem Internetu z elektronikami z całego Świata.

No dobrze, kończmy już tą wycieczkę w przyszłość i wracajmy do konkretów. Oto, czym chciałbym Was zainteresować na kolejnych wykładach BASCOM College (są to tylko główne tematy wykładów):



Wykład 4
Emulacja sprzętowa i programowa. Z listów wiem, ze napotykacie tu na poważne problemy i dlatego chciałbym możliwie szeroko omówić ten temat.

Wykład 5
Magistrala I2C. To temat rzeka, dotyczący jednego z najważniejszych elementów większości systemów mikroprocesorowych. Trudny do realizacji w językach niskiego poziomu, a banalnie prosty w MCS BASIC.

Wykład 6
Komunikacja 1WIRE. „Magiczne tabletki DALLAS", termometry cyfrowe, układy zdalnego sterowania, zamki szyfrowe, nieulotne pamięci, wszystko to połączone ze sobą za pomocą zaledwie .... jednego przewodu sygnałowego!

Wykład 7
BASCOM, a assembler. Omówienie możliwości jakie daje łączenie programu napisanego w MCS BASIC z wstawkami napisanymi w asemblerze, a nawet pisanie całych programów w tym języku i kompilowanie ich w środowisku BASCOM'a.

Pamiętajcie, Drodzy Czytelnicy, że są to tylko tematy główne wykładów. Tematy dodatkowe zależą także od Was, od propozycji i uwag jakie znajdą się w mojej skrzynce e-mail.

Zbigniew Raabe,
e-mail: zbigniew.raabe@edw.com.pl